OOA = Object Oriented Analysis
OOD = Object Oriented Design

1. explain OOA process with diagram
2. What do you mean by domain analysis in OOAD? Differen betn OOA and OOD.
3. What do you mean by inheritance, encapsulation and inheritance? Explain how is message passing done?
4. What is domain analysis? Explain domain analysis process.
5. What are different steps involved in it?
6. What do you understand by object-oriented pardigm? What are the steps in identifying the elements an object model for management of object oriented software projects?
7. Describe the concept of information hiding w.r.t software design in your own words.
8. What are class, object, attributes and methods? Explain with example.
9. Translation of OOA to OOD model.

Last Software

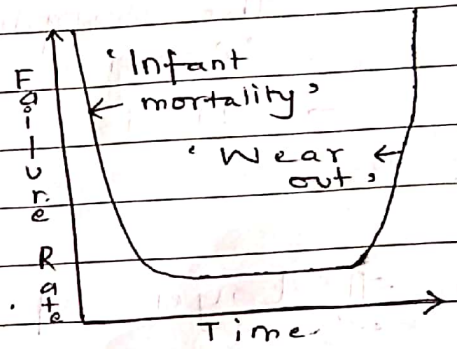
Software Project Management And Concepts

Characteristics of software.

1. Software is developed or engineered; it is not manufactured.
2. doesnt wear out
3. custom built

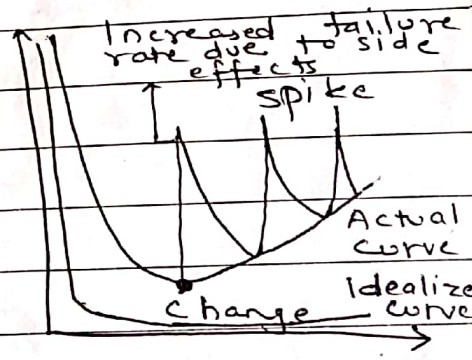
1. Software is developed
   - In both activities, high quality is achieved through good design.
   - manufacturing phase for hardware can introduce quality problems that are nonexistent.
   - for software defect can be easily corrected.

2. Figure depicts failure rate as function of the time for hardware.
   - It is 'bathtub curve'.
   - Hardware exhibits relatively high failure rates early in its life.



In software

- The failure rate curve for software should take the form of the 'idealized curve'.
- Software doesnot tear out but it does deteriorate.



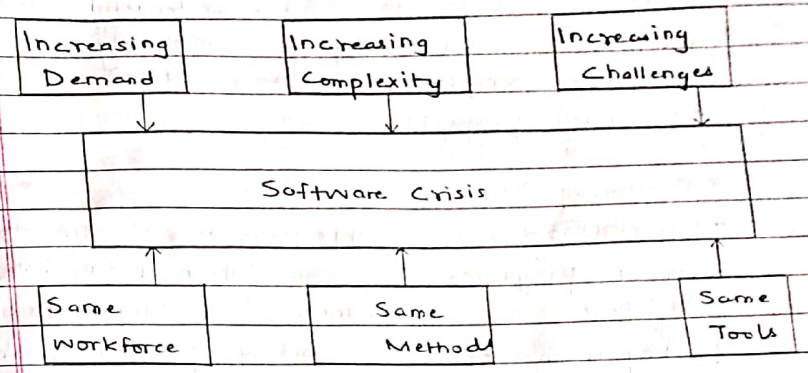
- During its life software will undergo change.
- When changes are made, errors will be introduce causing the failure rate curve to spike.
- Before the curve can return to the original steady-state failure rate, another change is requested, causes the spike again.

3. • In the hardware world, component reuse is a natural part of.
• A software component should be designed and implemented so that it can be reused in many different programs.

Software Engineering
→ The application of a systematic, disciplined, quantifiable approach to the development, operation and maintainence of software; that is the application of engineering to software 'According to IEEE'
→ It is a layered technology.

Layers of Software Engineering
- Tools → Automated or semi automated support
- Methods → Analysis, design, code, testing
- Process → Collection of Activities, actions
- A quality focus → quality of software

Software Crisis
• Software crisis difficulty of writing useful and efficient computer programs in the required time
• Using same workforce, same methods, same tools even though rapidly increasing in software demand, complexity of software and software challenges.
• Increase in the complexity of software, many software problems ~~arouse~~ arose because existing methods were insufficient.



Software Myths (important 5 or 7 mark) developer-2, management-2, customer-2.
→ Software myths - erroneous beliefs about software and the process that is used to built it
→ Myths have a number of attributes that make them insidious.
* Management Myths
→ We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?
• Reality: • Use of books?
• Existence of books?
• Reflects modern software engineering practise?
• Complete?
→ If we get behind schedule, we can add more programmers and catch.
• Software process is not manufacturing.
• people who were working must spend time educating the newcomers.
• People can be added but only in a planned and well coordinated manner.

→ If I decide to outsource the software project to a third party, I can just relax and let that firm build it
• If an organization doesnot understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

* Customer Myths

→ A general statement of objectives is sufficient to begin writing programs - we can fill in detail later
• Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster.
• Unambiguous requirements are developed only through effective and continuous comb^n betn customer and developer.

→ Software requirements continually change, but change can be easily accomodated because software is flexible.
• software requirements change, but the impact of changes varies with the time at which it is introduced.
• when requirements changes are requested cost impact is relatively small
• as time passes, the cost impact grows rapidly.

* Practitioner's myth

→ once we write the program and get it to work, our job is done.
• "the sooner you begin 'writing code' the longer it will take you to get done"
• 60 and 80 % of all effort expended on software will be expended after it is delievered to the customer for the first time.

→ until I get the program "running" I have no way of accessing its quality.
• quality assurance mechanisms can be applied from the inception of a project.
• technical review.
• Software reviews (testing for finding certain classes of software defects)

→ The only deliverable work product for a successful project is the working program.
• work products (Models, Documents, Plans, provide a foundation for successful engineering and, more important guidance for software support).

→ Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.
• not about creating documents.
• creating a quality product.
• Better quality leads to reduced rework.

C	P	M	C	D
• Communication	• Planning	• Model	• Construction	• Deployment

Generic Process Framework

- Communication

~~Planning~~ 1. Before any technical work commence, it is imp to communicate and collaborate with the customer and other stake holder.

2. The intent is to understand stakeholder's objectives for the project and to gather requirements that help define software and functions.

- Planning

1. It helps to define the software project plan.
2. Software Engineering plan defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work product to be produced, and a work schedule

- Modelling

1. By creating the models helps to better understand the software requirement and the design that will achieve those requirement.

- Construction

1. The activity combines code generation and the testing that is required to uncover errors in the code.

- Deployment

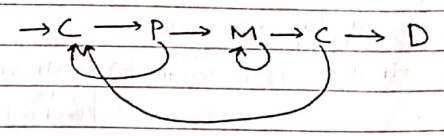
1. The software is delievered to the customer who evaluates the delievered product and provides feedback based on the evaluation.

Process Flow

A) → C → P → M → C → D

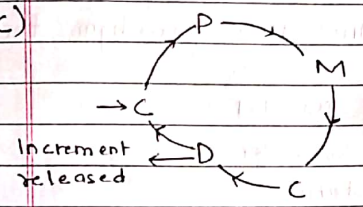
A) Linear Process Flow

B)



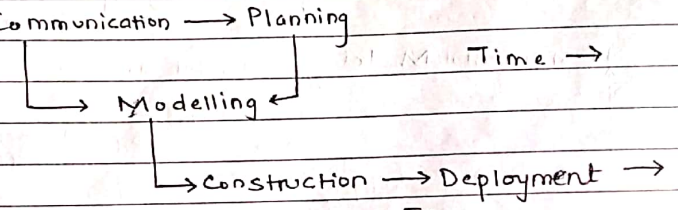
B) Iterative Process Flow

C)



C) Evolutionary Process Flow

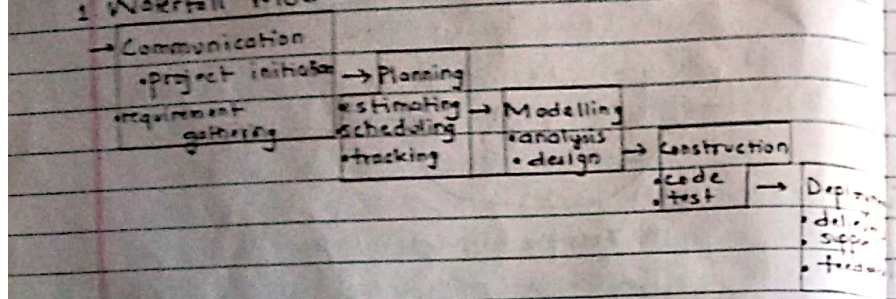
D)



D) Parallel Process Flow

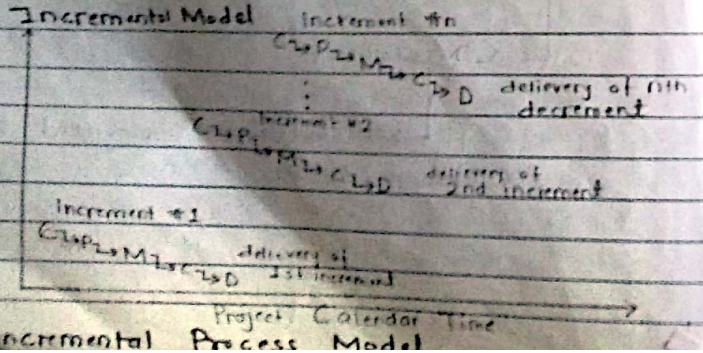
* Process Model
1. Waterfall Model (Linear Model)
2. Increment Process Model
3. Evolutionary Process Model (Prototyping, The Spiral
4. Rapid Application Model (RAD)
- Agile Process
- Unified Process

1 Waterfall Model



- The waterfall model is the oldest paradigm for software engineering
- When to use Waterfall Model?
  a. Requirement are very well known
  b. Production defn is stable
  c. Technology is understood
  d. New version of existing product.

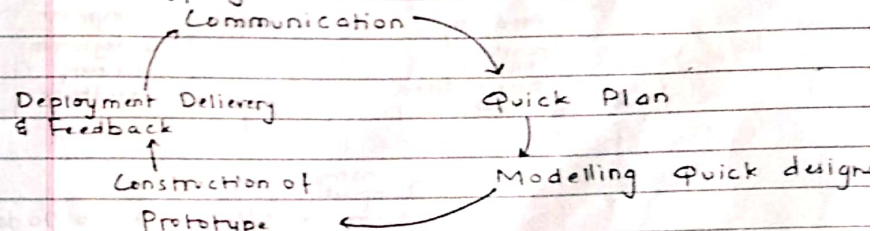
2. Incremental Model



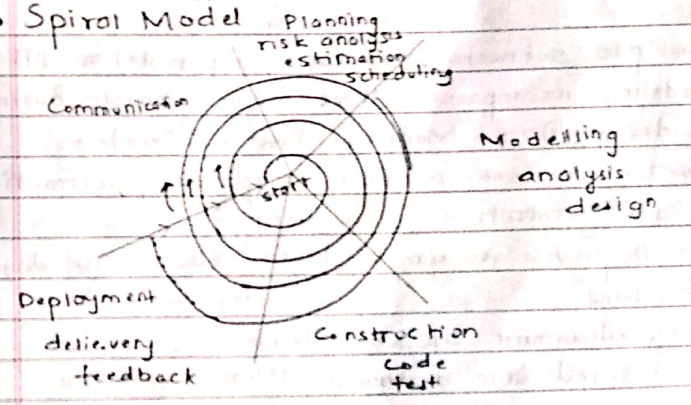
- Incremental Process Model
- Incremental model delivers a series of releases, called increments, that provide progressively more functionality for customers as each increment delivers
- The first increment is often a core product
- Basic requirements are addressed but many supplementary features remain undelivered

When to use?
- Useful when staffing unavailable for a complete implementation of project till deadline.
- If the core product is well received, then additional staff (if required) can be added to implement the next increment.

3 Evolutionary Model
- Prototyping Model



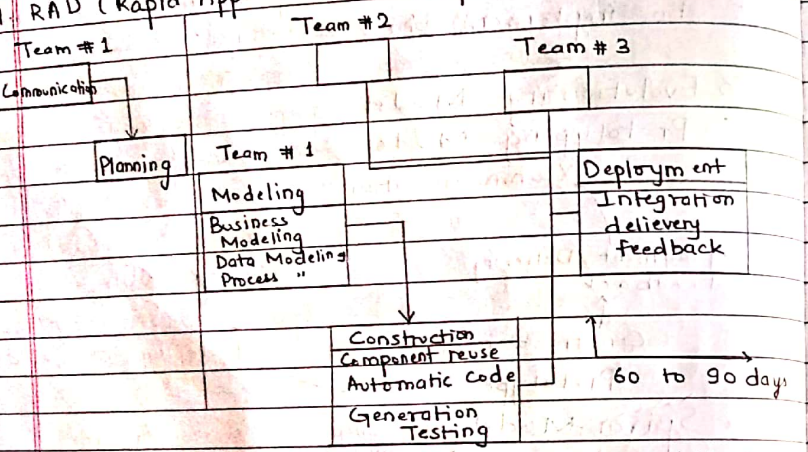
- Spiral Model



- is the combo of waterfall & ... with ...
- Each phase in spiral model begins with ... ends with client reviewing
- S/W is developed in a series of incremental series

- The spiral model is a risk-driven process model.
- Software is developed in series of evolutionary releases.
- During early iterations, release might be a model or prototype.
- Later iterations, increasingly complete versions of the product.
- The spiral model is the realistic approach to the development of large scale system and software.

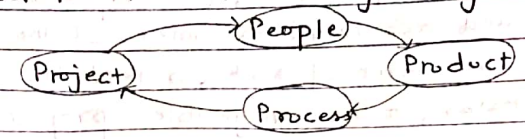
4. RAD (Rapid Application Development Model)



- Multiple software team work in parallel on diff functi...
- Modeling encompasses three major phases: Business Modeling, Data Modeling, Process Modeling
- Construction uses reusable components, automatic codes generation and testing.
- Overall process is completed in 60 to 90 days.

* Problems
- required number of RAD team.
- not suited when technical teams are weak.
- requires modularity.

Four P's in Software Engineering



1. People
2. Product
3. Process
4. Project

1. People :
- People are the primary key factor for the success of the any organization.
- Software Engineering has developed a people maturity model to accomplish its strategic business objectives.
- For successful software production – organization environme... must perform
  - Good Working Environment
  - Staffing
  - Performance management
  - Training
  - Compensation

- The players → Managers, Practitioners, Customers, End Us...
- Software Team
- Team Leaders

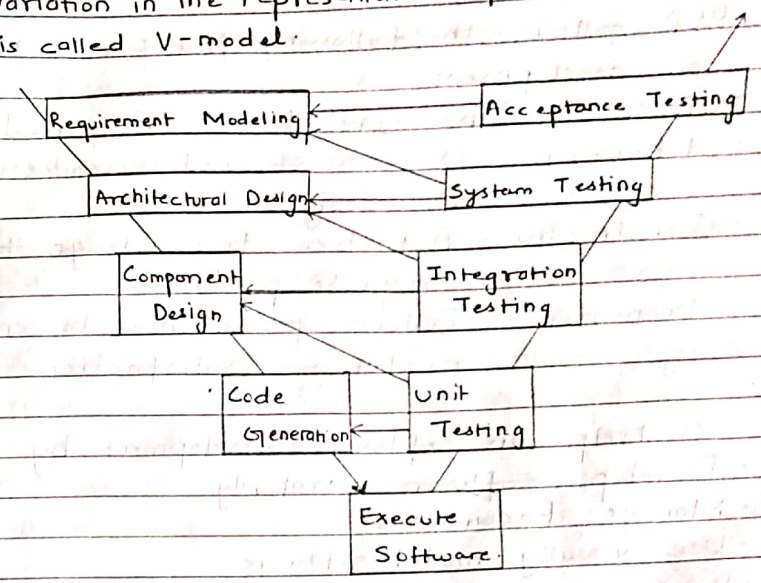
2. Product
- before a project can be planned, product objectives and scope should be established.
- Alternative solutions should be considered and technical and management constraints, ~~it is impossible to define~~ should be defined.

- Without product information, it is impossible to determine reasonable (and accurate) estimates of the cost.
- an efficient assesment of risk, a realistic breakdown of project tasks, or a manageable project schedule that provides a meaningful indication of progress.

3. Process
- Software process provides the framework from which a comprehensive plan for software development can be established.
- Detail operation to do for the project development may differ but the general framework for software development remain same.
- We can use any process to develop a software:
→ Linear Sequential Model
→ Prototype Model
→ RAD Model
→ Increment Model
→ Spiral Model
→ Concurrent development Model

4. Project
- We conduct, planned and controlled software projects to manage complexity of project.
- To avoid project failure, a software project manager and the software engineers who built the product must.
  → avoid a set of common warning signs.
  → understand ~~that~~ the critical success factors that leads to good project management.
  → develop a common ~~source~~ sense approach for planning, monitoring, and controlling the project

5. V-Model
- Variation in the representation of the waterfall model is called V-model.



ASSIGNMENT - 1

Qn.1. Describe the Rational Unified Process (RUP) for software development.

→ Rational Unified Process (RUP) is a software application development technique with many tools to assist in coding the final product & tasks related to this goal. RUP is an object oriented approach used to ensure effective project management & high quality software production.

- It has following key characteristics
* Use - Case driven from inception to deployment
* Architecture - Centric where architecture is a function of user needs.
* Iterative and Incremental, where large

projects are divided into smoller projects.

RUP follows the following four phase roles for development:

- Inception: The core idea is envisioned.
- Elaboration: Uses cases and architecture are designed.
- Construction: Activities from design to complete product.
- Transition: Follows up activities to ensure customer satisfaction.

It helps in software development by:

- Develop software iteratively
- Manage Requirements
- Use visually model software
- Verify software quality
- Control change to software.
- Use component based architectures

2 What do you mean by agility? Describe the agile process for software development.

→ Agile is a time bound, iterative approach software ~~development~~ delievery that builds software incrementally from the start of the project instead of trying to deliever all at once. Thus agility means effective (rapid & adaptive) response to change, effective communication among all stakeholder to force the team to focus on software rather than design and documentation using iterative process for delievering the working model of software quickly to the customer.

It helps in software development by

→ Deploying the software quicker and thus it increases the trust of computer

→ By better adaptation to rapidly changing requirements and respond routes

→ Helps in getting immediate feedback which help to improve the software in next increment

→ Continuous alternation to technical existence and good design

3 Differentiate between waterfall model and incremental model.

→

	Waterfall Model	Incremental Model
1	Need of detail documentation.	Detail documentation is not really necessary.
2	High risk of failure of project	Low risk of failure of project
3	Flexibility to make changes is difficult	Flexibility to make changes in it easy
4	There is long wasting time for running software in it.	There is short wasting time for running software in it
5	Large team is required.	No need of large team.

CamScanner

	Waterfall Model	Incremental Model
6.	Testing is done in it after completion of all coding phase. (Customer evaluation after final stage)	Testing is done in it after every iteration of phase (Customer evaluation in each increment)
7.	Only one cycle in it.	Multiple development cycles take place in it.
8.	Returning to previous phase is not possible.	Returning to previous phase is possible.
9.	Overlapping of phase is not possible.	Overlapping of phase is possible.
10.	This model is used for implementing the software product when the requirement is clearly defined.	When there is always a possibility of changing the requirement, we prefer this model.

Chapter : 2

Software Metrics

* Measure : A measure provides a quantitative indicati of the extent, amount, dimension, capacity, or size of some attribute of a product or process

* Metric : Metric as a 'quantitative measure of the degree to which a system, component, or process possesses a given attribute.

* Indicator : An indicator is a metric or condition of the metrics that provides insight into the software process, a software project, or the product itself.

- Metrics strongly ~~software~~ support software project management activities.
- They relate to the four functions of management as follows :
  - Planning
  - Organizing
  - Controlling
  - Improving

* Size Estimation Metrics
- Size of a program is not the number of bytes that the source code occupies.
- It is not the size of the executable code.
- It is an indicator of the effort and time required to develop the program.
- Size of program indicates development complexity.

- Estimating the problem size is fundamental to estimating the effort, time, and cost of planned software.

* Software Measurement
- Direct measure and indirect measure.
- Direct measures include cost and effort applied in the development of software.
- Direct measures of the software include how many lines of code (LOC) produced, execution speed, memory size and defects reported.
- Indirect measures include

Includes
- Functionality
- Complexity
- Efficiency
- Reliability
- Maintainability

* Lines of Code
- The simplest among all metrics available to estimate project size.
- Project size estimated by counting the no. of source instructions.
- Lines used for commenting, header lines ignored.
- To find LOC at the beginning of a project divide module into sub modules and so on until size of each module can be predicted.

* Disadvantages of LOC
- Gives a numerical value of problem size that vary widely with individual coding styles

```
if (x>y)                x>y ? x++ : y++ ;
  then x++ ;
else
    y++ ;
```

- Effort needed for analysis, design, coding, testing etc (not just coding).
- Larger Code Size → Better Quality?
- Logical Complexity?
  Complex Logic → More Effort
  Simple Logic → Less Effort.

```
while (i < 4)                printf ("testing");
{ printf ("testing");        printf ("testing")
  }                          "      "
                             "      "
```

- Accurate computation of LOC only after project completion.

V. Imp

Example of LOC

- Estimated line of code = 33,200
- Productivity = 620 LOC/PM = Person Month
- Labour Rate = $8000/PM
- Cost ~~line~~ per line of code = ? = $\frac{8000}{620}$ = $13
- Estimated Project Cost = ? = 33200 × 13 = $431,600
- Estimated Labor Effort = ? = 33200 / 620 = 54 PM

- Cost of per line of code = $\frac{\text{Labour Rate}}{\text{Productivity}}$

- Estimated project cost = Estimated line of code × cost per line of code.

- Estimated laboureffort = $\frac{\text{Estimated line of code}}{\text{Productivity}}$

$\frac{\text{Estimated KLOC}}{\text{Productivity}}$

Function Point Metric

- Size of software product computed directly from problem specification.
- Size of software = no. of different functions/features it supports.

Input → (Function) → Output

- Many features → Larger Size.
- Apart from that, size depend on
  - no. of files
  - no. of interfaces
  - no. of enquiries.

Functional Point.

- Size of Function Point (FP) = Weighted sum of those five problems characteristics.

1. No. of inputs : Data items input by user. (Group of users inputs taken together)
   → Employee Account
   - Name
   - Age
   - Sex
   - Address
   - Acc Name
   - Acc Number
   - Acc Open Date.

2. No. of outputs : Reports, Screen, outputs

3. No. of inquires : Interactive queries made by user.

4. No. of files : Logical files. Eg: data structures, physical files

5. No. of interfaces : Interfaces for exchanging info. Eg: disk, tapes, communication links.

Measurement Parameter	Count		Weighting factor Simple	Avg	Complex		
No. of user inputs		×	3	4	6	=	
No. of user outputs		×	4	5	7	=	
No. of user inquires		×	3	4	6	=	
No. of files		×	7	10	15	=	
No. of external interfaces		×	5	7	10	=	
Count total						=	

Information domain value	S_{opt}	S_m Likely	S_{pess}	Est Count	Avg Weight	FP count
No. of external i/p	20	24	30	24	4	97
No. of external o/p	12	15	22	16	5	78
No. of external inquiries	16	22	28	22	4	88
No. of internal logical files	4	4	5	4	10	42
No. of external interface files	2	2	3	2	7	15
Count Total						320

- Expected value for estimate variable size (s) = ?
- Make optimistic, most likely, pessimistic, estimate for each item, then compute expected value.

1. $S = \frac{S_{opt} + 4 S_m + S_{pess}}{6}$

$= \frac{20 + 4 \times 24 + 30}{6}$

$= 24.33$

$\Rightarrow 24.33 \times 4 = S \times \text{Weight}$

$= 97.33$

2. $S = \frac{12 + 4 \times 15 + 22}{6} \times 5$

$= 78.33$

$= 78$

3. $S = \frac{16 + 4 \times 22 + 28}{6} \times 4$

$= 88$

4. $S = \frac{4 + 4 \times 4 + 5}{6} \times 10$

$= 41.666$

$= 42$

5. $S = \frac{2 + 4 \times 2 + 3}{6} \times 7$

$= 15.16$

$= 15$

FP estimate $= \text{Count total} \times [0.65 + 0.01 \times \Sigma(f_i)]$ (Value adjustment factor)

$= 320 \times [0.65 + 0.01 \times 52]$

$= 375$

# Factor	Value
Backup and recovery	4
Data Communications	2
Distbd processing	0
Performance critical	4
Existing operation environment	3
Online data entry	4
I/p transaction over multiple screens	5
Master files updated online	3
Info domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion / Installation in design	3
Multiple Installation	5
App designed for change	5
Value adjustment factor	1.17

Pu. 2015

Compute the function point value for a project with following information domain characteristics.

No. of user input = 32
" " " o/p = 60
" " " inquiries = 24
No. of files = 8
No. of external interface = 2

Answer;

No. of i/p = 32×4 = 128
No. of user o/p = 60×5 = 300
No. of inquiries = 24 × 4 = 96
No. of file = 8 × 10 = 80
No. of external interface = 2×7 = 14
count Total = 618

FP estimate = Count total * [0.65 + 0.01 × 52]

= .375

Organizational productivity = 6.5 FP/PM

Labour Rate = $8000

Cost per FP = $8000 / 6.5
= $1230

[Since our project is average we use the value 1.17.]

Pn. You are required to develop student management system (MIS) for NCIT in which estimated lines of codes (LOC) is calculated to be 75,000 and a average productivity, for this system is 250 LOC/PM and the labor rate is 8,500 per month. Calculate.

1. Estimate project cost
2. Estimated Effort.

Soln;

Estimated LOC = 75,000
Avg Productivity = 250 LOC/PM
Labor Rate = Rs. 8500 PM
Cost per line of code = 8500 / 250
= 34

Estimate project cost = 8500 / 250 × 75000
= Rs. 2,550,000

Now;

Estimated Effort = LOC / Productivity
= 75,000 / 250
= 300 PM

Question,

Given, No. of i/p = 50
No. of o/p = 40
No. of inquiries = 35
No. of files = 6
No. of interface = 4

Typical Characteristics with
0 = No influence
1 = Incidental
2 = Moderate
3 = Average
4 = Significant
5 = Essential

Step 1 :

Measurement Parameter	Count	Wting Factor (Avg)	Total
1. No. of i/p	50	4	200
2. No. of o/p	40	5	200
3. No. of inquiries	35	4	140
4. No. of files	6	10	60
5. No. of interface	4	7	28

Function point can be computed as,
Functional Point (FP) = UFP × VAF
where, UFP = Unadjusted Functional Point
VAF = Value Adjustment Factor

Step 2 :
UFP = 200 + 200 + 140 + 60 + 28
= 628

VAF = (0.65 + (0.01 × (14 × 3)))
= 1.07

Step 3 :
FP = 628 × 1.07
= 672

∴ FP estimated = 672

Productivity = 6.5 FP / PM
Labor Rate = $8000

(A) Cost per FP = $\frac{\$8000}{6.5}$
= $1230

(B) Effort = $\frac{\text{FP Estimate}}{\text{Productivity}}$
= $\frac{672}{6.5}$
= 53 PM

(C) Total project Cost = 672 × 1230
= $826560

Qn. You develop a software (MIS) for NCIT student management system in which the ~~estimated lines~~ ~~of codes (LOC)~~ is ~~calculated to be 35,000~~ ~~and a review of the historical data~~
No. of ip = 98
No. of o/p = 51
No. of inquiries = 47
No. of external interfaces = 32
No. of files = 61
Assuming that the complexity of given MIS develop is average, compute the function point if the productivity of S/W Developers is 35 FP/PM and their salary is Rs. 15000 per month average estimate cost of software.

Ans: Productivity = 35 ~~PM~~ FP/PM
Labor Rate = Rs. 15000

Step 1 Parameters	Count	Weighting Factor (Avg)	Total
1 No of i/p	98	4	392
2 No of o/p	51	5	255
3 No of inquiries	47	4	188
4 No of files	61	10	610
5 No of interface	32	7	224
			1669

Functional point can be computed as,

$$FP = UFP \times VAF$$
$$= 1669 \times 1.07$$
$$= 1785.83$$
$$= 1786$$

Step 2:

$$\text{Cost per FP} = \frac{15,000}{35} = \frac{\text{Labor rate}}{\text{productivity}}$$
$$= 42.8$$
$$= 43$$

$$\text{Effort} = \frac{\text{FP estimate}}{\text{Productivity}}$$
$$= \frac{1786}{35}$$
$$= 51$$

Total project cost $= 1786 \times 43$
$= 76,798$

PU 2012

	Info domain value	S_{opt} Optimistic	S_m Likely	S_{pes} Pessimistic
1.	No of i/p	20	24	30
2.	No of o/p	12	15	22
3	No of inquiries	16	22	28
4	No of files	4	4	5
5	No of External interface	2	2	3

Given all complexity adjustment values are average. The organizational average productivity for system of this type is 6.5 FP/PM and labor rate of \$8000 per month. Also calculate the total cost required and effort required

Ans → 1. $S = \frac{S_{opt} + 4S_m + S_{pes}}{6}$

$$= \frac{20 + 4 \times 24 + 30}{6}$$
$$= 24.33$$
$$= 24.33 \times 4 = 97.33$$

2. $S = \frac{12 + 4 \times 15 + 22}{6} \times 5$

$$= 78.33$$
$$= 78$$

3. $S = \frac{16 + 4 \times 22 + 28}{6} \times 4$

$$= 88$$

4. $S = \frac{4 + 4 \times 4 + 5}{6} \times 10 = 42$

5. $S = \frac{2 + 4 \times 2 + 3}{6} \times 7 = 15$

$\therefore$ Count total $= 320$.

$FP = \text{Count total} \times [0.65 + 0.01 \times 52]$
$= 37.5$

Productivity = 6.5 FP/PM
Labor Rate = \$8000 PM.
Total Cost = ?
Effort = ?

$$\text{Cost per F.P} = \frac{\text{Labour Rate}}{\text{Productivity}}$$
$$= \frac{8000}{6.5}$$
$$= \$1230$$

$$\text{Total Cost} = \text{FP} \times \text{Cost per F.P}$$
$$= 375 \times 1230$$
$$= \$461250$$

$$\text{Effort} = \frac{\text{FP}}{\text{Productivity}}$$
$$= \frac{375}{6.5}$$
$$= 57.69 \text{ PM}$$

Categories of Metrics
- Product Metrics
- Process Metrics
- Project Metrics

- Product Metrics.
  ↳ It describes the characteristics of the product, such as size, complexity, performance, efficiency.
- Process Metrics
  ↳ It describes the effectiveness and quality of the process. Example:
  - Effort required in the process.
  - Time to produce the product.
  - Number of defects found during testing.
- Project Metrics
  ↳ It describes the project characteristics and execution
  Eg: • Number of software developer.
  - Staffing pattern over the life cycle of the software
  - Cost and Schedule
  - Productivity.

Attribute of Effective Software Metrics
1. Simple and Computable
2. Empirically and Intuitively Persuasive (Satisfy Engineer's Intuitive)
3. Consistent and Objective
4. Consistent in the use of units and dimensions
5. Programming Language Independent
6. An Effective Mechanism for High Quality feedback.

Metric for Software Quality [Important from book]
- Measuring Quality
  → Indicators to measure the quality:
  1. Correctness
  2. Maintainability
  3. Integrity (loyally functioning or not)
  4. Usability (user friendly or not)

Defect Removal Efficiency

↳ A quality metric that provides benefit at both the project and process level is detect removal efficiency (DRE)

$$DRE = \frac{E}{E+D}$$ where, E = Errors found before delievery of the software.

D = Defects found after the delievery

Metric for Small Organization : See Book

$$DRE = \frac{E_{change}}{E_{change} + D_{change}}$$

Project Estimation Technique

- Emprical estimation technique
- Heuristic technique
- Analytic estimation technique

- Empirical estimation techique : making an educated guess of the project using past experiences. Eg Delphi and Expert judges

- Heuristic Technique : Project parameter can be modeled by the mathematical expression.

- Analytic Estimation Technique : Like Heuristic Technique but based on scientific facts.

COCOMO

↳ proposed by Barrey W. Boehm in 1981.

↳ algorithmic software estimation model in the world

↳ Predicts the effort and duration of a project based on :

- inputs relating to the size of the resulting systems and
- number of 'cost drives' that effect productivity

Development Modes

1. Organic Mode
   - Relatively Simple,
   - Small projects with a small team are handled

2. Semidetached Mode
   - For intermediate software projects (little complex compared to organic mode projects in terms of size
   - have a mix of rigid and less than rigid requirement

3. Embedded Mode
   - Project to be developed within a tight set of hardware and software operational constraints
   - Example of complex project : Air traffic control system

COCOMO Model

- The coefficients of a, b, c, d for the three modes are :
  - Types of COCOMO Model
    - 1 → Basic COCOMO Model
    - 2 → Intermediate COCOMO Model
    - 3 → Detailed COCOMO Model

1. can be used for quick & slightly rough calc of s/w cost
   It isn't that accurate
2. takes these cost into control / account
3. Works on more detailed factors

Basic COCOMO

- Effort = $a(KLOC)^b$ PM
- Time = $c(\text{Effort})^d$ Months
- No. of people required = (Effort Applied) / (Development Time)

COCOMO Model

Software Project	Effort a	b	Schedule c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Basic COCOMO

- Example: The size of organic software is estimated to be 32,000 LOC. The avg salary for software engineering is Rs. 15000/- per month. What will be effort and time for the completion of the project?

Soln:

Effort applied = $2.4 \times (32)^{1.05}$ ~~1.05~~ PM = 91.33 PM

(Since: 32000 LOC = 32 KLOC)

Time = $2.5 \times (91.33)^{0.38}$ ~~0.38~~ Month = 13.899 Months

Cost = Time × Avg salary per month

= 13.899 × 15000

= Rs. 208480.85

People required = Effort applied / Development time

= 6.57

= 7 person

1. Effort = $a[KLOC]^b$ PM

$= 2.4\left[\frac{32,000}{1000} \text{ KLOC}\right]^{1.05}$ PM

= 91.33 PM

2. Time = $c[\text{Effort}]^d$ Months

$= 2.5 \times [91.33]^{0.38}$

= 13.899 Months

3. No. of people required = $\frac{\text{Effort Applied}}{\text{Development Time}}$

$= \frac{91.33}{13.899} = 6.57$

= 7 persons

Qn. example: . . . embedded

1. Effort = $3.6 \times \left(\frac{32000}{1000}\right)^{1.20}$

= 230.4 PM

2. Time = $2.5 \times [230.4]^{0.32}$

= 14.2539 months

3. No. of people required = $\frac{230.4}{14.2539}$

= 16.1639

Cocomo (Constructive Cost Model) is a model based on Lines of Codes (LOC). It is a cost estimate model for software projects and often used as a process of ~~predecting~~ predicting the various things associated with making a project such as size, effort, cost, time and Quality.

No of people 10

C, mulatchel

effort = 19 b

Time = 19

* Software Risk
- Anticipated unfavourable event
- When risk turns to reality, it hampers successful and timely completion of project.

Anticipate / Identify risks — Contingency Plans

Employee — Give incentive to employee

Risky Component — Out source third party

Categories of risk:
ⓐ Project Risk (project plan → cost increase)
ⓑ Technical Risk (quality and time
ⓒ Business Risk
ⓓ Known Risk (can be unconvered after careful evaluation)
ⓔ Predictable Risk (from past projects)
ⓕ Unpredictable Risk (extremely difficult to identify)

Basic Characteristics of Risk:
- Uncertainity: Risk may or may not happen
- Loss: If risk becomes a reality losses will occur.

ⓒ # Top 5 Business Risk are;
- Market risk: system that no one really wants.
- Strategic risk: product no longer fits into the overall business strategy.
- Sales risk: the sales force doesn't understand how to sell the product.
- Management risk: losing the support of senior management.
- Budget risks: losing budgetary or personnel commitment.

* Reactive and Proactive Risks
- Reactive Risks
→ Software engineer corrects a problem as it occurs or never worrying about problems until they happened. This is often called a fire-fighting mode.

- Proactive Risks
→ A proactive strategy ~~becomes~~ begins long before technical work is initiated where potential risks are identified, their probability and impact are assessed, and they are ranked by importance.
→ Then, the software team establishes a plan for managing risk. The primary objective is to avoid risk, but all risks cannot be avoided so excercise to make them minimal.

Risk Management Process

- Risk identification
  - Identify project, product and business risks

- Risk Analysis
  - Assess the likelihood and consequences of these risks.

- Risk Planning
  - Draw up plans to avoid or minimize the effects of the risk.

- Risk Monitoring
  - Monitor the risks throughout the project

Risk Identification :

- The different categories of risk (project, technical, business, known, predictable and unpredictable) can be further divided as:
  a) Generic risks :
  b) Product - specific risks :

- Risk Item checklist
  1. Product Size
  2. Business impact
  3. Process definition
  4. Development environment
  5. Technology to be built
  6. Staff size and experience.

Risk assessment

- Rank risk in terms of their damage causing potential
- The overall risk exposure, RE

  Risk Exposure (RE) = r × c

- Here, r is probability of occurence for a risk, and c is cost to project should risk occur.

Case Study

PU 2017

- For example, assume that software team defines a project risk in as follows :
  - Risk Identification : Only 70% of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.
  - Risk Probability : 80% (likely)
  - Risk Impact : 60 reusable software components were planned. If only 70% can be used, 18 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development). Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is $14.00.

Soln;

Overall cost (impact) to develop the components would be = 18 × 100 × 14

= $25,200

Risk Exposure (RE) = r × c

= 0.8 × 25,200

= $20,200

Risk Analysis

→ Access probability and effect of each risk.
→ Probability maybe very low, low, moderate, high or very high.
→ Risk effects might be catastrophic, serious, tolerable or insignificant.

Risk Planning

→ Consider each side risk and develop a strategy to manage that risk.
→ Reactive vs proactive approach
→ Avoidance Strategies
  • The probability that the risk will arise is reduced.
→ Minimization Strategies
  • The impact of the risk on the project or product will be reduced.
→ Contingency plans
  • If risk arises, contingency plans are to deal with that risk.
  • Risk mitigation, risk transfer

Risk Monitoring

→ Access each identified risks regularly to decide whether or not it is becoming less or more probable.
→ Also access whether the effects of the risk have changed.
→ Each key risk should be discussed at management process meetings.

Q. Adding people to a late software project makes it later 'Identify the risk and develop risk information sheet.

Risk Refinement

→ During early stages of project planning, a risk maybe stated quite generals

RMMM

→ Risk Mitigation, Monitoring and Management.
→ Assessment helps to develop strategy.
→ An effective strategy must consider three risk avoidance, risk monitoring and risk management and contingency planning.

783 pg

RMMM plan

→ Risk management, monitoring and managemen plan.
→ Some software team don't use a format RMMM plan and each risk is individually documented using a risk information sheet.

Risk Information Sheet

RMM Plan

- Risk mitigation

Risk Table

Risks	Category	Prob	Impact
Size estimate maybe significantly low	PS	60%	2
Larger no. of users than planned	PS	30%	3
Less reuse than planned	PS	70%	2
End-users resist system	BU	40%	3
Delivery deadline will be lightened	BU	50%	2
Funding will be lost	CU	40%	1
Customer will change requirements	PS	80%	2
Technology will not meet expectations	TE	30%	1
lack of training on tools	DE	80%	3
Staff inexperienced	ST	30%	2
Staff turnover will be high	ST	60%	2

- Impact Values
  - 1 - catastrophic
  - 2 - critical
  - 3 - marginal
  - 4 - negligible
- Risk Category
  - PS - Project Size Risk
  - BU - Business risk
  - TE - Technology risk
  - DE - Development risk
  - ST - Stakeholder risk
  - CU - Customer risk

Chapter 4: Software Quality Assurance

Quality

- It is a complex and multifaced concept that can be described from 5 diff POV:

1. Transcendental View:
   Something that immediately recognizes, but cannot explicitly define
2. User View:
   If a product meets an end user's specific goals, it exhibits quality.
3. Manufacturer's View:
   Product confirms to the original specification
4. Product View:
   Quality can be tied in inherent characteristics (Eg functions and features) of a product.
5. Value-based View:
   How much a customer is willing to pay for a product?

Software Quality

- An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.

The definition serves to emphasize three important points :

- Effective software process :
  establishes the infrastructure that supports any effort at building a high-quality software product.

- Useful product :
  delivers the content, functions and features that the end users desires in a reliable error-free way.

- Added measurable values :
  For producer :- high-quality software requires less maintenance effort, fewer bug fixes, and reduced customer support
  For end user :
  1. greater software produce revenue
  2. better profitability when an application supports a business process, and/or
  3. improved availability of information that is crucial or vital for the business.

Quality Factors :

- The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identified six key attributes.

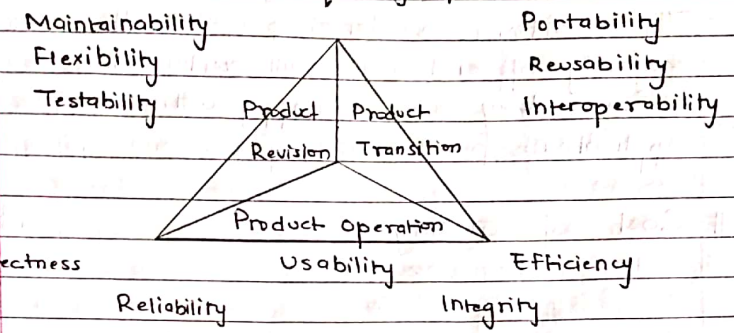
ISO/IEC 9126

Six Key Attributes

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Pg: 417

McCall's software quality factors



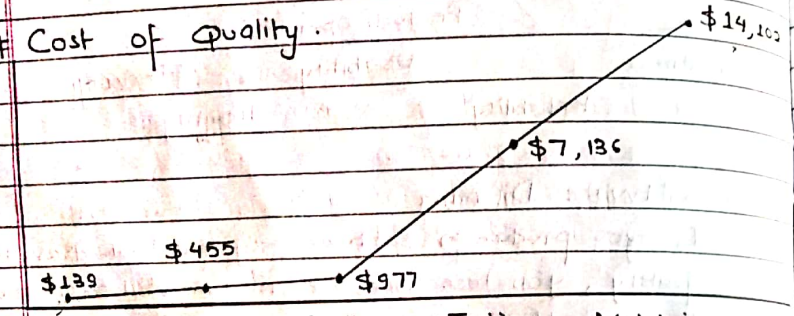
Software Dilemma

→ If you produce a software system that has terribl quality, you lose because no one will want to buy it. If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anywa Either you missed the market window, or you simply exhausted all your resources. So people in industry try to get that magical middle ground where the product is good enough not to be rejected right away, such as during evaluation, but also not the object of so much perfectonism and so much work that it would take too long or cost too much to complete.

Good Enough Software

- Good Enough Software delivers high-quality functions and features that users desires, but at the same time it delivers other more obscure or specialized functions and features that contain known bugs.
- The software vendor hopes that the vast majority of end users will overlook the bugs because they are so happy with other application functionality.

Cost of Quality.



contd:

Qn. 2017 Spring

2b) Why are software reviews important? What are the guidelines for conducting FTR?

contd: The relative costs to find and repair an error or defect increase dramatically as we go from prevention to detection to internal failure to external failure costs.

- The average industry costs to correct a defect during code generation is approximately $977 per error discovered during system testing is $7,136 per error.

- The cost of quality can be divided into costs associated with prevention, appraisal and failure.

→ Prevention costs include

1. The cost of management activities required to plan and coordinate all quality control and quality assurance activities.
2. The cost of added technical activities to develop complete requirements and design models
3. test planning costs, and
4. the cost of all training associated with these activities.

- Appraisal costs include activities to gain insight into product condition the 'first time through' each process.

Examples of appraisal costs include:

- Cost of conducting technical reviews product cost of data collection and metrics evaluation.
- Cost of testing and debugging.

- Failure Costs are those that would disappear if no errors appeared before or after shipping a product to customers.
- Failure costs maybe subdivided into internal failure costs and external failure costs.
- Internal failure costs are incurred ~~and~~ when you detect an error in a product prior to shipment.
- Internal failure costs include cost required to perform rework (repair) to correct an error

Failure Cost

Internal failure Cost

External failure cost

- External failure:
- This costs is associated with defects found after the product has been shipped to the customer.
- Examples of external failure costs are product return and replacement, help line support, and labor costs associated with warranty work.

Software Reviews

- Many types of reviews can be conducted as part of software engineering.
- An informal meeting around the coffee machine is a form of review, if technical problems are discussed.
- A formal presentation of software architecture to an audience of customers, management, and technical staff is also a review.
- by casual reviews, walkthroughs and inspection
- A technical review (TR) is the most effective filter from a quality control standpoint. Conducted by software engineers (and others) for software engineers, the TR is an effective means for uncovering errors and improving software quality.

Imp: 2018 Fall 2@

Formal Technical Review

→ FTR is a software quality control activity performed by software engineers (and others).

The objectives of an FTR are:

1. to uncover errors in function, logic.
2. To verify that the software under review meets its requirements.
3. to ensure that the software has been represented according to predefined standards;
4. to achieve software that is developed in a uniform manner; and
5. to make projects more manageable.

* The Review Meeting

Every review meeting should abide by the following constraints:

- Between three and five people (typically) should be involved in the review.
- Advance preparation should occur but should require no more than two hours of work for each person.
- The duration of the review meeting should less than two hours.
- FTR focuses on a specific (and small) part of the overall ~~sources~~ software.
- For eg: rather than attempting to review an entire design; walkthroughs are conducted for each component or a small group of component
- By narrowing the focus the FTR has higher ~~likely~~ ~~food~~ ~~of~~ likelihood of uncovering error.

Review Reporting and Record Keeping

- During the FTR, a reviewer actively records all issues that have been raised.
- These are summarized at the end of the review meeting and a review issues list is produced.
- In addition, a formal technical review summary report is completed. A review summary report answers 3 questions:

1. What is reviewed?
2. Who reviewed it?
3. What are the findings and conclusion?

Review Guidelines

1. Review the product, not the ~~procedure~~ producer.
2. Set an agenda and maintain it.
3. Limit debate and rebuttal. (खण्डन)
4. Enunciate problem areas, but don't attempt to solve every problem noted.
5. Take written notes.
6. Limit the no. of participants and insist upon advance preparation.
7. Develop a checklist for each product that is likely to be reviewed.
8. Allocate resources and schedule time for FTRs.
9. Conduct meaningful training for all reviewers.
10. Review your early views.

Elements of Software Quality Assurance

- Software quality assurance encomposses a broad range of concerns and activities that focus on the management of software quality.
- Standards
- Reviews and audits
- Testing
- Error / defect collection and analysis.
- Change management.
- Education
- Vendor Management
- Security Management / Safety Mngmt
- Risk Management
- ~~Sa~~

SQA task, Goals and Metric

SQA task

- Prepare an SQA plan for project.
- Participate in the development of the project's software process.
- Reviews software engineering activities to verify compliance with the defined software process.
- Audits designated software work product to verify compliance with those defined as part of the software process.
- Ensures that deviations in software work and work products are documented and handled according to a documented process.
- Records any noncompliance and reports to senior management.

Goals
- Requirements quality
- Code quality
- Quality control effectiveness.

Goal	Attribute	Metric
Requirement quality	Ambiguity	No. of ambiguous-modifiers (eg: many, large, humanfriendly)

Formal Approach to SQA
- There can be argued that a computer program is a mathematical object.
- A rigorous syntax and semantics can be defined for every programming language, and a rigorous approach to the specification of software requirements is available.
- If the requirements model (specification) and the programming language can be represented in a rigorous manner, it should be possible to apply mathematic proof of correctness to demonstrate that a program confirms exactly to its specifications.

Imp Page: 456 2019 Fall 3@

Statistical Software Quality Assurance
- Statistical quality assurance reflects a growing trend through out industry to become more quantitative about quality.
- For software, statistical quality assurance implies the following steps:

1. Information about software errors and defects is collected and categorized.
2. An attempt is made to trace eachother and defect to its underlying cause (eg: non-confirmance to specification, design error, violation of standards, poor communication with the customer).
3. Using the Pare to principle (80 percent of the defects can be traced to 20% of all possible causes), isolate the 20% (the vital few)
4. Once the vital few cause have been identified, move to correct the problems that have caused the errors and defects.

Six Sigma for Software Engineering
- Six Sigma is the most widely used strategy for statistical quality assurance in industry today.
- Six Sigma strategy is a rigorous and disciplined methodology.
- Uses data and statistical analysis to measure and improve a company's operational performance

The Six Sigma methodology defines three core steps:
- Define customer requirements and deliverables and project goals via well defined methods of customer communication.
- Measure the existing process and its output to determine current quality performance (collect defect metrics).
- Analyze defect metrics and determine the vital few causes.

Imp 2016 Fall 2b)

Software Reliability

- Software reliability is defined in statistical terms as 'the probability of failure-free operation of a computer programming, a specified environment for a specified time.
- To illustrate, program X is estimated to have a reliability of 0.999 over eight elapsed processing hours.
- In other words, if program X were to be executed 1000 times and require a total of eight hours of elapsed processing time (execution time), it is likely to operate correctly (without failure) 999 times.

Measure of Realiability and Availability

- There has been an ongoing debate over the relationship betn key concepts in hardware reliability and their applicability to software.
- Although an irrefutable link has yet to be established it is worthwhile to consider a few simple concepts that apply to both system elements.
- If we consider a computer-based system, a simple measure of reliability is meantime between-failure (MTBF):

$$MTBF = MTTF + MTTR$$

where the acronyms MTTF and MTTR are mean-time-to-failure and mean-time-to-repair respectively.

- Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as:

$$\text{Availability} = \frac{MTTF}{MTTF + MTTR} \times 100\%$$

Software Safety

- Software Safety is a software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail.
- If hazards can be identified early in the software process, software design features can be specified that will either eliminate or control potential hazards.
- Software safety is a

2016 PU Fall :
2b) Explain software reliability? Guidelines of FTR. (7)
Short notes on @ ISO Quality Standard.

The SQA Plan

- The SQA plan provides a road map for instituting software quality assurance.
- The standard for SQA plan has been set. The standard recommends a structure that identifies
  - → The purpose and scope of the plan.
  - → A description of all software engineering work product.
  - → All applicable standards and practises that are applied during the process.
  - → SQA actions and tasks and their placement through the software process.
  - → The tools and methods that supports SQA actions and tasks.
  - → Software configuration management procedures
  - → Methods for assembling safeguarding and maintaining all SQA-related records.

PU 2016 Fall

b) What is SQA? Discuss the activities involved as a part of SQA plan.

International Standard Organization (ISO)

- The ISO 9000 standards specifies the guidelines for maintaining a quality-system.
- ISO 9000 is a series of three standards : The types of software industries to which the different ISO standard apply are as follows :
- ~~This~~ 1. ISO 9001
  - → The standard applies to the organizations engaged in the design, development, production and servicing of goods.
  - → This is the standard that is applicable to most software development organizations.

2. ISO 9002

→ The standard applies to the organizations that do not design products but are only involved in production.
→ Example of this category of industries can be steel or car manufacturing industries who buy the product plant design from external sources and are involved in only manufacturing those products.
→ Therefore, ISO 9002 is generally not applicable to software development organization.

3. ISO 9003

→ This standard applies to organizations involved only in installation and testing of the products.

Why to get ISO Certification.

Some of the benefits that accrue (to increase) to organizations after obtaining the ISO certification.

- Confidence of customer in an organization is enhanced when organization has the ISO certification.
- ISO 9000 requires a well-documented software production process to be in place.
- ISO 9000 makes the development process focused, efficient and cost effective.
- ISO 9000 certification points out the weak points of an organizations and recommends remedial actions.

How to get ISO Certification?

- An organizations intending to obtain the ISO certifications applies to an ISO 9000 registar for registration.
- The ISO 9000 registration process consists process consists of following stages;
  - * Application Stage
  - * Pre-assessment
  - * Document review and adequacy audit
  - * Compliance audit.
  - * Registration
  - * Continued Surveillance

imp Chapter 5 : Software Configuration Management

Software Configuration Management

- Change is inevitable when computer software is built. (sure to happen)
- Change increases the level of confusion when you and other members of a software team are working on a project.
- Confusion arises when changes are not analyzed before they are made, recorded before they are implemented, reported to those with a need to know, or controlled in a manner that will improve quality and reduce error.
- The art of coordinating software development to minize confusion is called configuration management.
- Configuration management is the art of identifying, organizing and controlling modifications to the software being built by a programming team.
- The goal is to maximize productivity by minimizing mistakes.
- Software Configuration Management (SCM) is an umberella activity that is applied throughout the software process because change can occur at any time.

SCM activities are developed to

1. identify change,
2. control change,
3. ensure that change is being properly implemented
4. report changes to others who may have an interest.

Software support and SCM

- Support is a set of software engineering activities that occur after software has been delievered to the customers and put into operation.
- Software configuration management is a set of tracking and control activities that are initiated when a software engineering project begins and terminates only when the software is taken out of operation.

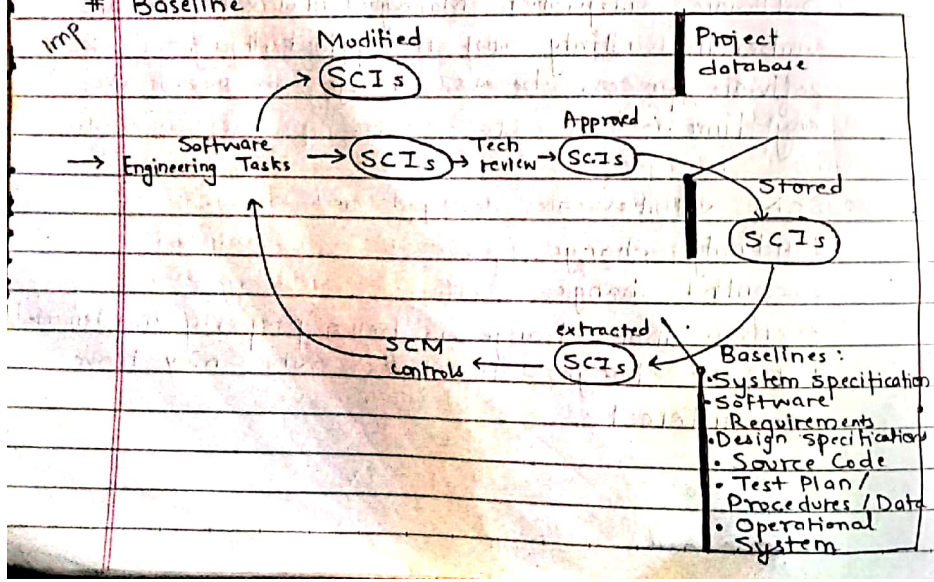
Elements of Configuration Management System

→ Four important elements that should exist when a configuration management system is developed.

- Component elements
- Process elements
- Construction elements
- Human elements.

Baseline

imp



PU 2019 Fall

What is SCM? Explain the role of baseline and SCI in SCM process with necessary diagram. [8]

- A baseline is a software configuration management concept that helps you to control change.
- Before a software configuration item becomes a baseline, changes maybe made quickly and informally
- However, once a baseline is established, changes can be made, but a specific, formal procedure must be applied to evaluate ~~and~~ in every each change.
- Software engineering tasks produce one or more SCIs
- After SCIs are reviewed and approved, they are placed in a project database.
- When a member of a software engineering team wants to make a modification to a baselined SCI, it is copied from the project db into the engineer's private workspace.

Software Configuration Item

- Software configuration item as information that is created as part of the software engineering process.
- SCI could be considered to be a single section of a large specification or one test case in a large suite of tests
- More realistically, an SCI is all or part of a work product (eg: a document, an entire suite of test cases, or a named program component).

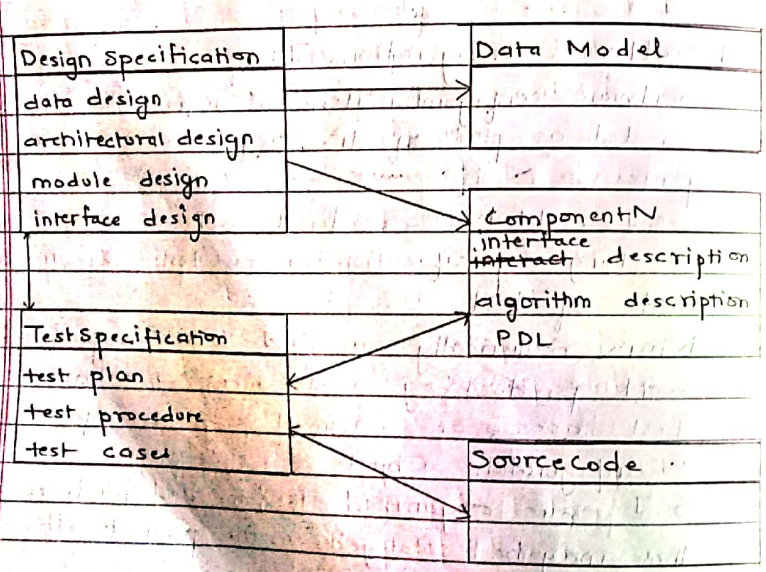
* Configuration Object

- SCIs are organized to form configuration objects that may be cataloged in the project db with a single name.
- A configuration object has a name, attributes, and is 'connected' to other objects by relationships.

- the configuration objects; Design Specification, DataModel, ComponentN, SourceCode, and TestSpecification are each defined separately.

Configuration Object

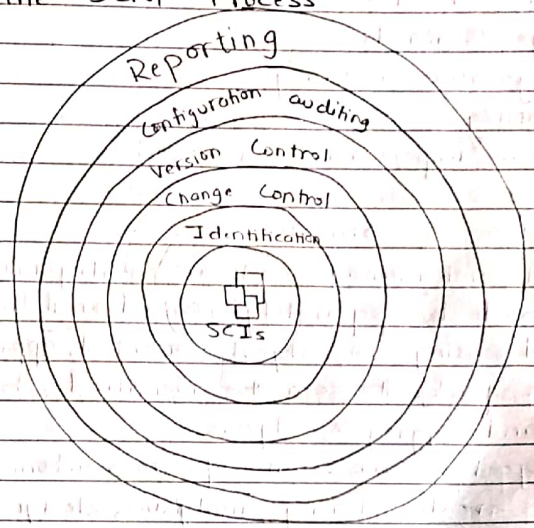
- object is related to the others as shown by the arrow.
- A curved arrow indicates a compositional relation. That is, DataModel and ComponentN are part of the object DesignSpecification.
- A double-headed straight arrow indicates an interrelationship.
- If a change were made to the SourceCode object, the inter-relationships enable you to determine what other objects (and SCIs) might be affected



SCM Features

- Versioning
- Dependency tracking and change management
- Requirements tracing
- Configuration management
- Audit trails

The SCM Process



SCM Process

The software configuration management process defines a series of tasks that have four primary objectives.

1. to identify all times that ~~are~~ collectively defin the software configuration.
2. to manage changes to one or more of these items.
3. to facilitate the construction of different version of an application,

4. to ensure that software quality is maintained as the configuration evolves over time.

SCM tasks

→ A process that achieves these objectives lead to definition of five SCM tasks, they are

- Identification
- Version Control
- Change Control
- Configuration auditing
- Reporting

*explanation important (From book)

Identification of objects

- To control and manage software configuration items, each should be separately named and then organized using an object-oriented approach.
- Two types of objects can be identified: basic objects and aggregate objects.
- basic object is ~~an~~ unit of ~~trans~~ information that you create during analysis, design, code or test.
- For example, a basic object might be a section of a requirements specification, part of a design model, source code for a component.
- An aggregate object is a collection of basic objects and other aggregate objects.
- For example, a DesignSpecification is an aggregate object.
- Each object has a set of distinct features that identify it uniquely: a name, a description, a list of resources and a 'realization'.
- configuration object identification can also consider the relationships that exist betn named objects
- For example, using the simple notation.
  class diagram <part-of> requirements model;
  Requirements model <part-of> requirements specification

SCM Repository

- Data Integration
- Methology enforcement
- Documend Standardization.

Version Control

- Version control combines procedures and tools to manage different ~~controls~~ versions of configuration objects that are created during the software process.
- A version control system implements or is directly integrated with four major capabilities:
  1. a project database
  2. a version management capability
  3. a make facility (collect all contgn object and construct specific version)
  4. issue tracking (bug tracking)

Pg. 597

Change Control

~~Need for change is required~~

Change Control Process I

need for change is required
↓
change request from user
↓
developer evaluates
↓
change report is generated
↓
change control authority decides

→ request is queued for action
→ change request is denied, user is informed

↓

Change control Process II

↓
assign people to SCIs (Software Configuration Items)
↓
check-out SCIs
↓
make the change
↓
review/audit the change
↓
establish a 'baseline' for testing
↓

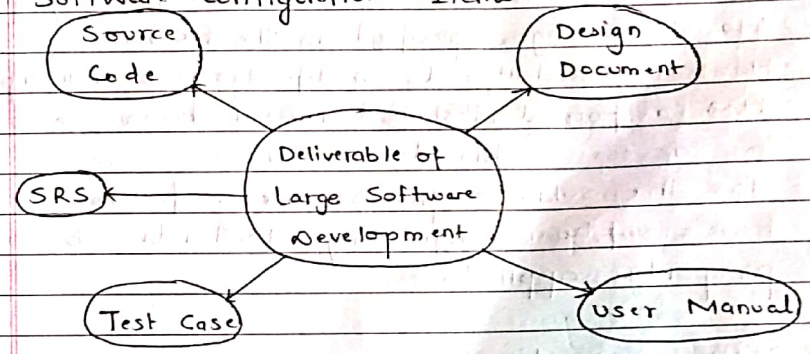
Change control Process III

Change control Process – III

↓
perform SQA and testing activities (Software Quality Assurance)
↓
check-in the changed SCIs
↓
promote SCI for inclusion in next release
↓
rebuild appropriate version
↓
review/audit the change
↓
include all changes in release

Software Configuration Items

imp



Change Control

- Change Control is vital. But the forces that make i necessary also make it annoying.
- We worry about change because a tiny perturbati (deviation) in the code can create a big failure in the product. But it can also fix a big failure or enable wonderful new capabilities.

- We worry about change because a single rogue developer could sink the project; yet brilliant ideas originate in the minds of those rogues, and a burdensome change control process could effectively discourage them from doing creative work.

To ensure that change has been properly implemented:

i. Formal Technical Reviews
ii. Software Configuration Audit.

Configuration Audit

→ The audit asks and answers the following questions:

- Has the change specified in the ECO been made?
- Have any additional modifications been incorporated?
- Has a formal technical review been conducted to assess technical correctness?
- Has the software process been followed and have software engineering standards been properly applied?

Status Reporting

- Configuration status reporting (status accounting) is an SCM task that answers the following questions:

i. What happened?
ii. Who did it?
iii. When did it happen?
iv. What else will be affected?

SCM Standard

- Over the past two decades a number of software configuration management standards have been proposed.
- SCM standards, such as MIL-STD-483, DODSTD-480A and MIL-STD-1521A focused on software developed for military applications.
- Recent ANSI/IEEE standards, such as ANSI/IEEE Stds. No. 828-1983, No. 1042-1987 and Std. No. 1028-1988 [IEE 94] are applicable for nonmilitary software and are recommended for both large and small software engineering organizations.

Chapter 7 - Testing

- According to ANSI/IEEE 1059 standard, testing can be defined as: A process of analyzing a software item to detect the differences betn existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.
- In simple words testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

Objectives

- ~~finding defects~~

Who does Testing?

- It depends on the process and the associated stakeholders of the project(s).
- Different companies have different designations for people who test the software.
- In most cases, the following professionals are involved.
  - Software Tester
  - Software Developer
  - Project Lead / Manager
  - End User.

Verification Vs Validation.

- Verification refers to the set of tasks that ensure that software correctly implements a specific function.
- Validation refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.
- Verification: "Are we building the product right?"
- Validation: "Are we building the right product?"
- Verification and validation includes a wide array of SQA activities:
  1. technical reviews, quality and configuration audit performance monitoring, simulation, feasibility study documentation review, db review, algorithm analysis, development testing, usability testing, qualification testing, acceptance testing and installation testing.

Debugging vs Testing

- Debugging is the process of fixing a bug in the software.
- It can defined as the identifying, analyzing and removing errors.
- Testing is the process to find bugs and errors.
- Debugging is the process to correct the bugs found during testing.

Testing Methods

- Black Box Testing
- White Box Testing
- Gray Box Testing

imp

Black Box Testing

- Testing without having any knowledge of the interior working of the application.
- Tester doesn't have the access of source code
- Needn't know the internal working of application.
- Main focus in the functionality of the system as a whole.
- This is known as the behaviour testing.

* Methods of BBT
  - → Graph Based Testing Methods
  - → Equivalence ~~Partio~~ Partitioning
  - → Boundry Value Analysis
  - → Orthogonal Array Testing

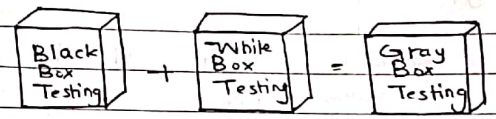
White Box Testing

- Internal details and system is made visible.
- White Box technique is used by both developers as well as testers.
- Highly efficient method to find bugs.
- Known as Glass Box, open Box, Transparent Box Testing.
- Main focus on the structural part of the application.

* Methods of WBT
  - → Basis Path Testing
  - → Control Structure Testing

Gray Box Testing

- Gray-box testing is a combⁿ of white-box testing and black-box testing.
- The aim of this testing is to search for the defects.
- Gray-box testing is beneficial.



* Methods of GBT
  - → Matrix Testing
  - → Regression Testing
  - → Pattern Testing

Software Testing Levels

- Functional
- Non-Functional

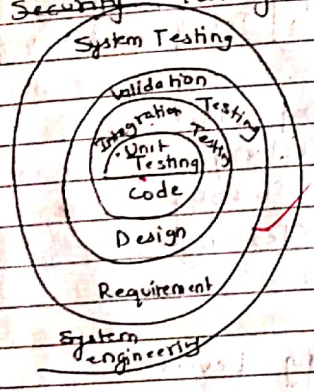
Functional Testing Types

1 → Unit Testing
2 → Integration Testing
3 → System "
4 → Regression "
5 → Acceptance "
6 → Alpha "
7 → Beta "

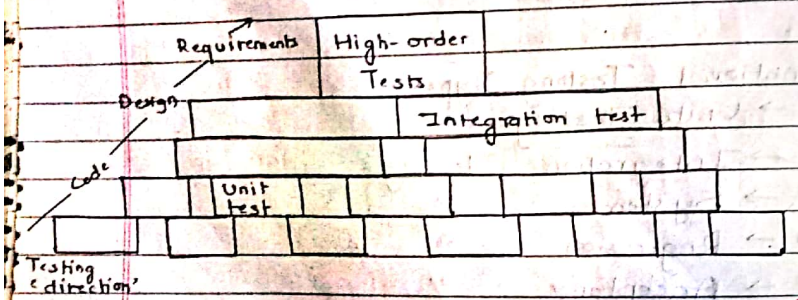
Non Functional Types

1 → Performance Testing
2 → Load "
3 → Stress "
4 → Usability "
5 → Profitability "
6 → Security "

Software ~~Security~~ Testing Strategy.



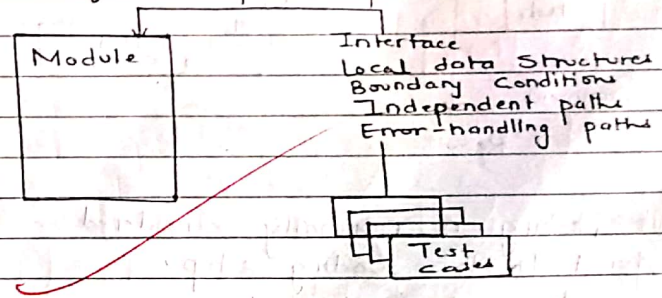
Software Testing Steps



- Unit testing makes heavy use of testing techniques to ensure the maximum error detection
- Components must be assembled or integrated to form the complete software packages
- Integration testing addresses the issues associated with the dual problems of verification and program construction.
- ~~Components must be ass~~ After the software has been integrated (constructed), a set of high-order tests is conducted.

Unit Testing

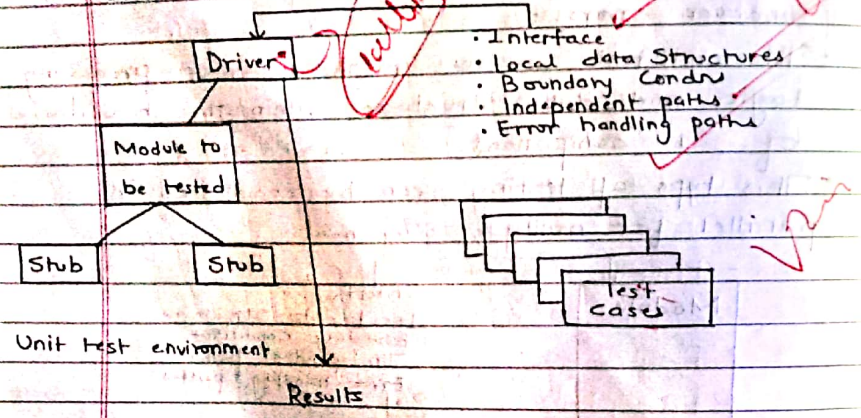
- Unit testing focuses verification effort on the smallest unit of software-design : software component or module.
- Using the component-level design description as a guide, important control paths are tested to uncover errors.
- The unit test focuses on the internal processing logic and data structures within the boundaries of a component.
- This type of testing can be conducted in parallel for multiple components.



Unit Test Consideration

- The module interface is tested to ensure that information properly flows into and out of the program unit under test.
- The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
- Boundary condns are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
- Independent paths (basic paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once.
- All error handling paths are tested.

Unit Test Procedure



- Interface
- Local data Structures
- Boundary Condns
- Independent paths
- Error handling paths

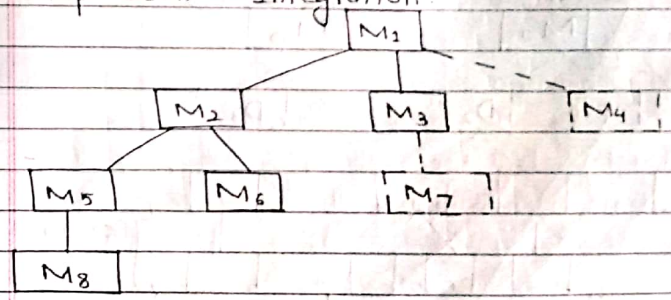
- Unit testing is normally considered as an adjunct to the coding step.
- The design of unit tests can occur before coding begins or after source code has been generated.

- A review of design information provides guidance for establishing test cases that are likely to uncover errors.
- Each test case should be coupled with a set of expected results.
- In most applications a driver is nothing more than a 'main program' that accepts test case data, passes such data to the component, and prints relevant results.
- Stubs serve to replace modules that are subordinate (invoked by) the component to be tested.
- A stub or 'dummy subprogram' uses the subordinate module's interface, may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing.

Integration testing

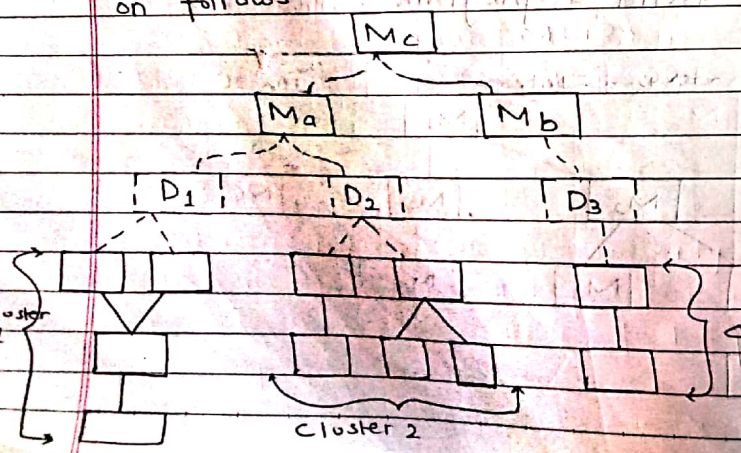
- Integration testing is a systematic technique for constructing the software architecture.
- The objective is to take unit-tested components and build a program structure.

Top-down Integration



Top-down Integration

- This testing is an incremental approach to construction of the software architecture.
- Modules are integrated by moving downward through the control hierarchy, beginning with the main control module.
- Modules subordinate to the main control module are incorporated ~~with~~ into the structure in either a depth-first or breadth-first manner.
- For example; selecting the left-hand path, components M1, M2, M5 would be integrated first.
- Next, M8 or (if necessary for proper functioning of M2) M6 would be integrated.
- Breadth-first integration incorporates all components directly subordinate at each level, moving across the structure horizontally.
- From the figure, components M2, M3 and M4 would be integrated first.
- The next control level M5, M6 and so on follows.



Bottom-up integration.

- This testing begins construction and testing with atomic modules (i.e components at the lowest levels in the program structure).
- the functionality provided by components subordinate to a given level is always available and the need for stubs is eliminated.

Regression Testing.

- Each time a new module is added as part of integration testing, the software changes.
- New data flow paths are established, new I/O may occur, and new control logic is invoked.
- These changes may cause problems with functions that previously worked flawlessly.
- In the context of an ~~tes~~ integration test strategy, regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

→ Regression test ensure changes have not propagated unintended side effect.

Smoke Testing.

- Smoke testing is an integration testing approach that is commonly used with product software is developed.
- the smoke-testing approach encompasses the following activities.

1. Software components that have been translated into code are integrated into a build. A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions

Smoke Testing

→ build kind of software Testing performed after to ascertain that the critical functionalities of the

Program are working fine

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. The integration approach maybe top down or bottom up.

- Benefits of Smoke Testing (from Book)
  - Integration risk is minimized
  - Progress is easier to access
  - Quality of end product is improved.
  - Error diagnosis and correction are simplified

System Testing Vo Imp

1. Recovery Testing
   - It is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed
   - If recovery is automatic re-initialization, check pointing mechanisms, data recovery and restart are evaluated for correctness
   - If recovery requires human intervention, the mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.

2. Security Testing
   - Security testing attempts to verify that protection mechanisms that protection mechanisms built into a system will, in fact, protect it from improper penetration.
   - During security testing, the tester plays the role(s) of the individual who desires to penetrate the system.

3. Stress Testing
   - Stress tests are designed to confront programs with abnormal situations.
   - Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency or volume. For example,
     - (a) special tests maybe designed that generate ten interrupts per second, when one or two is the average rate,
     - (b) input data rates may be increased by an order of magnitude to determine how i/p functions will respond.
     - (c) test cases that require max memory or other resources are executed.
     - (d) test cases that may cause thrashing in a virtual operating system are designed.
     - (e) test cases that may cause excessive hunting for disk-resident data are created.

4. Performance Testing
   - It is designed to test the run-time performance of software within the context of an integrated system.
   - It occurs throughout all steps in the testing process.
   - Performance tests are often coupled with stress testing and usually require both hardware and software instrumentation.
   - It is often necessary to measure resource utilization (eg: process or cycles) in an exacting fashion.

5. Deployment Testing

- In many cases, software must execute on a variety of platforms and under more than one operating system environment.
- Deployment testing, sometimes called configuration testing, excercises the software in each environment in which it is to operate.
- In addition, deployment testing examines all installation procedures and specialized installation software (e.g. 'installers') that will be used by customers, and all documentation that will be used to introduce the software to end users.

Test Case Design Template

Testcase Template

Modules	Test Case ID	Test Type	Priority	Test Scenario	Pre-Condn	Test Steps	Actual Result	Expired Result	Result P/F

Black Box Testing

* Boundary Value Analysis

Example 1:

Age : [ ] (accepts 18 to 56)

Boundary Value Analysis		
Invalid	Valid	Invalid
(min-1)	(min, +min, -max, max)	(max+1)
17	18, 19, 55, 56	57

Valid Test Cases

Enter the value 18
" " " 19
" " " 55
" " " 56

Invalid Test Cases

Enter the value 17
" " " 57

Example 2:

Name : [ ] (accepts 6-12 characters)

~~Bour~~

Boundary Value Analysis		
Invalid	Valid	Invalid
(min-1)	(min, +min, -max, max)	(max+1)
5	6, 7, 11, 12	13

Valid Test Cases

Enter the no. of character - 6
" " " " " - 7
" " " " " - 11
- 12

Invalid Test Cases

Enter the no. of char - 5
" " " " " - 13

Equivalence ~~Partioting~~ Partitioning

Example 1.

Age: [ ] (accepts 18 to 56)

Equivalence Partitioning		
Invalid	Valid	Invalid
<=17	18-56	>=57

Orthogonal Array Testing (Black Box)

- The orthogonal array testing method is particularly useful in finding region faults - an error category associated with faulty logic within a software component.

~~Contro~~

White Box

Control Structure Testing

Condition Testing :

- E1 < relational operator > E2
- where E1 and E2 are arithmetic expressions and <relational-operator) is one of following

Data Flow

- The data flow testing method selects test paths of a program according to the locations of definitions and uses of variables in the program.

Loop Testing

1. Simple loops
2. Nested loops
3. Concatenated loops
4. Unstructured loops

Validation Testing

- VT can be defined in many ways, but a simple definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer.
- Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing, etc.

Validation Test Criteria

- A test plan outlines the classes of tests to be conducted, and a test procedure defines specific test cases.
- After each validation test case has been conducted, one of two possible condition exists:

1) The function or performance characteristics confirms to specification and is accepted or
2) a deviation from specification is uncovered and a deficiency list is created.

Configuration Review

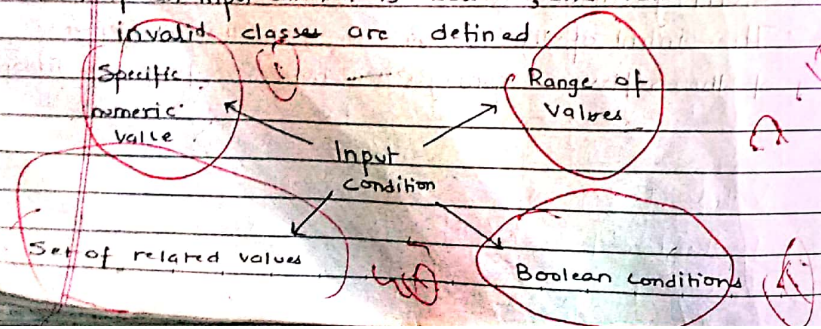
- The intent of the review is to ensure that all elements of the software configuration have been properly developed

Important

Equivalent partitioning

- Equivalence partitioning is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived.
- An ideal test case single-handedly uncovers a class of errors that might otherwise require many test cases to be executed before the general error is observed.
- Test-case design for equivalence partioting is based on evaluation of equivalence classes for an input condition.
- An input condition is either a specific numeric value, a range of values, a set of related values, or a Boolean condition.
- Equivalence classes maybe defined according to the following guidelines:

1. If an input condn specifies a range, one valid and two invalid equivalence classes are defined.
2. If an i/p condn requires a specific value, one valid and two invalid equivalence classes are defined.
3. If an input condn specifies a member of a set, ~~of~~ one valid and one invalid equivalence classes are defined.
4. If an input condition is Boolean, one valid and one invalid classes are defined.



Basis path testing method

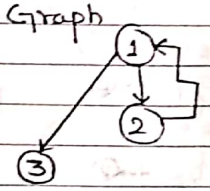
The following steps can be applied to derive the basis set:

1. Using the design or code as a foundation, draw a corresponding flow graph.
2. Determine the cyclomatic complexity of the resultant flow graph.
3. Determine a basis set of linearly independent paths.
4. Prepare test cases that will force execution of each path in the basis set.

While loop

```
While (x < y)
{ Print ("the value"); }
```

Flow of Graph



Cyclomatic Complexity

- It is a source code complexity measurement that is being corelated to a no. of coding errors.
- It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

The cyclomatic complexity V(G) can be calculated as:

V(G) = regions

V(G) = predicate nodes (P) + 1

V(G) = No. of edge (E) - No. of nodes (N) + 2

3. # Linearly Independent Path

Path 1 = 1 - 2 - 1 - 3

Path 2 = 1 - 3

4. Prepare test cases

For path 1 = 1 - 2 - 1

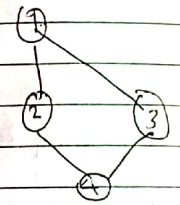
A test case is x = 5 and y = 7

For path 2 = 1 - 3

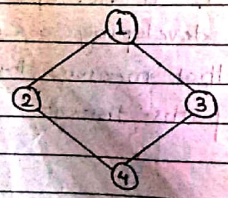
A test case is x = 7 and Y = 5

If Else

```
if (x>y)          → 1
{
  printf("Success");  → 2
}
else
{ printf("Fail"); }  → 3
                  → 4
```



i)



ii) Cyclomatic Complexity

V(G) = 2 regions

V(G) = predicate nodes + 1 = 1+1 = 2

V(G) = 4 edges - 4 Nodes + 2 = 2

iii) Linearly Independent Paths

- Path 1 : 1 - 2 - 4
- Path 2 : 1 - 3 - 4

iv) Prepare test cases

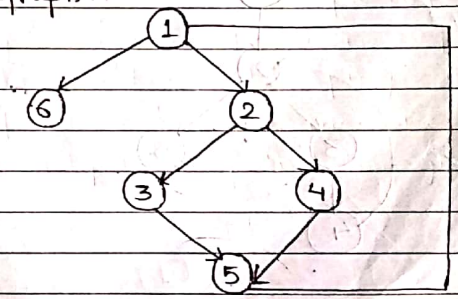
- Path 1 (x = 5, y = 2)
- Path 2 (x = 3, y = 5)
- ~~Path 3 (x = 2, y = 5)~~

#

```
while (x<y)        — 1
{  if (x==3)       — 2
     {             } — 3
  else             — 4
     {             } — 5
}                  — 6
```

```
if (  ) — 2
{
  if (  ) — 3
  {
  }
  else
  { — 4
  }
}
— 5
```

i) Flow Graph.



ii) Cyclomatic Complexity

V(G) = 3 regions

V(G) = Predicate nodes + 1 = 2 + 1 = 3

V(G) = 7 edges − 6 Nodes + 2 = 3

iii) Independent Path

Path 1 : 1 − 2 − 3 − 5 − 1 − 6

Path 2 : 1 − 2 − 4 − 5 − 1 − 6

Path 3 : 1 − 6

iv) Test cases

Path 1 : 1 − 6

x = 4 and y = 3

2 : 1 − 2 − 3 − 5 − 1 − 6

x = 3

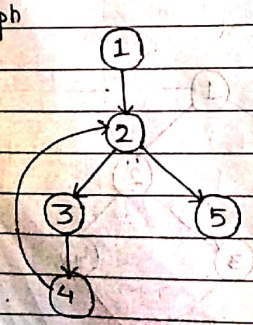
3 : 1 − 2 − 3 − 5 − 1 − 6

x = 2 or any other except 3

Qn. Find the cyclomatic complexity :

```
For (i=0 ; i<10 ; i++)
{ printf("Print the value of i") ; }
```

Step 1 : Flow Graph



Step 2 : Cyclomatic Complexity

V(G) = 2 regions

V(G) = Predicate nodes + 1 = 1 + 1 = 2

V(G) = 5 edges − 5 nodes + 2 = 2

Step 3 : Independent Path

Path 1 : 1 − 2 − 5

Path 2 : 1 − 2 − 3 − 4 − 2

Step 4 : Test Cases

Path 1 : 1 − 2 − 5

i = 12

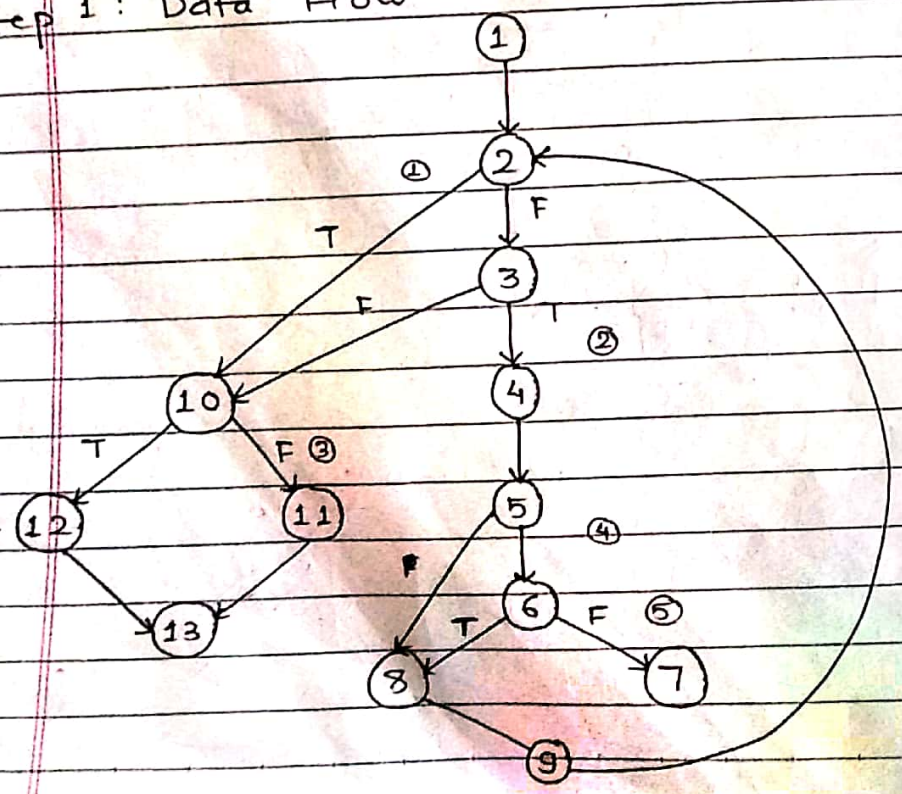
Path 2 : 1 − 2 − 3 − 4 − 2

i = 1

```
for( i= 0, i<10, i++)
{
  print("    ");
}
```

```
Pn.  int i = 1
     total.input = total.valid = 0 ;   → (1)
     sum = 0                           → (2)
(2) ← DO WHILE value[i] <- 999 AND total.input < 100
        increment total.input by 1 ;  → (4)
(5) ← If value[i] >= minimum AND value[i] <= maximum  → (6)
        THEN increment total.valid by 1 ;
             sum = sum + value[i] ;   → (7)
        Else SKIP
        END IF                        → (8)
        increment i by 1 ;
     END DO                           → (9)
     If total.valid > 10              → (10)
     THEN average = sum / total.valid ;  → (11)
        Else average = - 999 ;        → (12)
        END IF                        → (13)
     END . average.
```

Step 1 : Data Flow

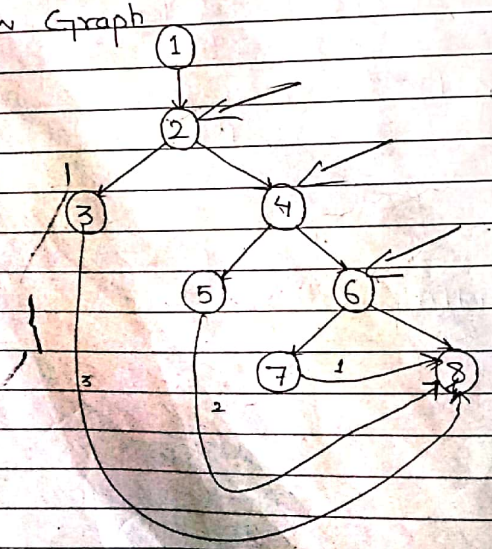


PU 2016

Qn

```
int a,b,c;                          } ①
d=b*b-4*a*c;                        — ②
if (d<0)
{ Real = - b/(2*a);
  d = - d;
  Num=pow(d, 0.5)                   } — ③
  Imag = num/(2*a);
}
else if (d==0).                     — ④
{ root1 = - b/(2*a); } — ⑤
  root2 = root1;
}
else if (d>0)                       — ⑥
{ root1 = (-b+sqrt(d)) / 2*a;       } — ⑦
  root2 = (-b -sqrt(d)) / 2*a;
}                                   — ⑧
```

Step 1: Flow Graph



Step 2: Cyclomatic Complexity

V(G) = 4 regions

V(G) = Predicate Nodes + 1 = 3 + 1 = 4

V(G) = 10 edges - 8 nodes + 2 = 4

Step 3: Independent Path

Path 1 : 1 - 2 - 3 - 8

Path 2 : 1 - 2 - 4 - 5 - 8

Path 3 : 1 - 2 - 4 - 6 - 8

Path 4 : 1 - 2 - 4 - 6 - 7 - 8

Step 4: Test cases

For path 1 :

Alpha and Beta Testing

- The alpha test is conducted at the developer's site by a representative group of end users
- The β-test is conducted at one or more end-user site
- Unlike α testing, the developer generally is not present. Therefore, β test is a 'live' application of the software in an environment that cannot be controlled by the developer.

Not testing chapter.

Identifying the elements of an object

1. Identify classes and objects.
2. Specifying the attributes
3. Defining Operations
4. Finalizing the object definition.

* Class Diagram

	Class Name
Attribute	
Methods	

* Make for student class

Student
id
Roll
name
set_stdinfo()
get_stdinfo()

Management of object-oriented software projects

- Better time to market
- Improved quality
- Greater resilience to change
- Increased degree of re-use.
- Complex system is divided into different subsystem

Object-Oriented Analysis

- The Booch method
- The Rumbaugh method
- The Jacobson method.

UML (Unified Modelling language)

- UML is the standard language for designing and documenting the system in a object oriented manner.
- It's a language where technical architecture can communicate with developers.
- It has 13 different types of UML.

UML

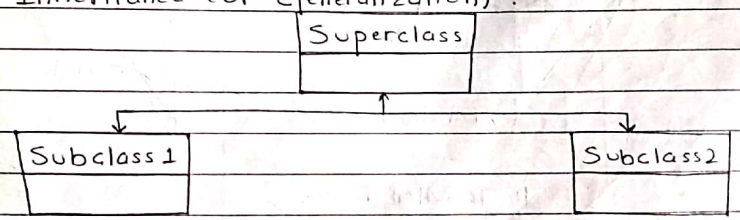
- User model view
- Structural model view
- Behaviour model view
- Implementation model view
- Environment model view.

Class Notation

- It consists of three parts :
  - Class Name
  - Class Attributes
  - Class Operations (Methods)

Class Relationships

- Inheritance (or Generalization) :-



- Aggregation
- Composition
- Dependency

Simple Association

Relationship between object

Player 1 : Player		Mongoose : Bat
Sport : Cricket	use →	Weight : 1.25kg
Gender : Male		Height : 78cm
Age : 23		Width : 2 inches

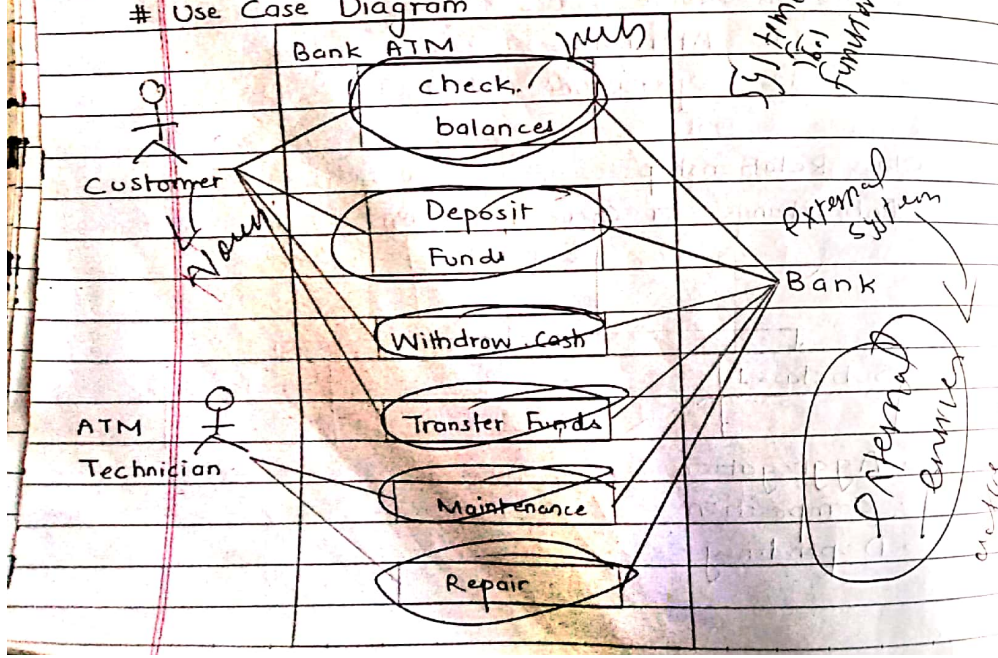
Figure : an object diagram using a dependency relationship

Object Name : Classifier Name
Attributes

Use Case

- Use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system.

Use Case Diagram



Class Responsibility Collaborator (CRC) card

Example :

Class Name	
Responsibilities	Collaborators
Student	
Student number	Seminar
name	
address	
enroll in a seminar	
drop a seminar	
request transcripts	

Object Oriented Analysis

→ During the object-oriented analysis, there is an emphasis on finding and describing the objects in the problem domain.

→ For example, in the case of the fligh information system, some of the objects include plane, flight and pilot.

Object Oriented Design

→ During the object oriented design there is an emphasis on defining software objects and how they collabrate to fulfill the requirements.

→ For example, a plane software object may have tail Number attribute and getFlightHistory method.

Domain Analysis

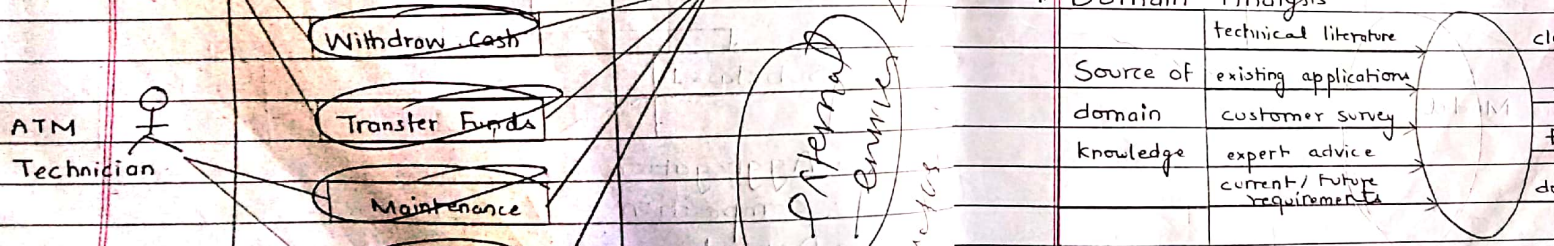
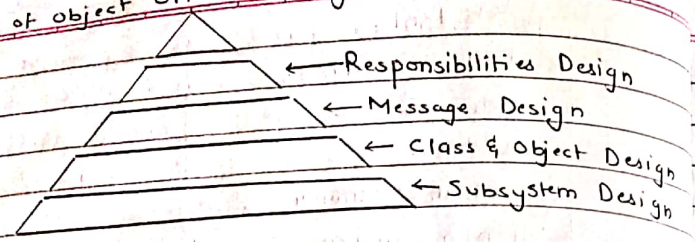


Fig: input and output for domain analysis

Ch 10

Four layers of object oriented Design Pyramid :



Message layers

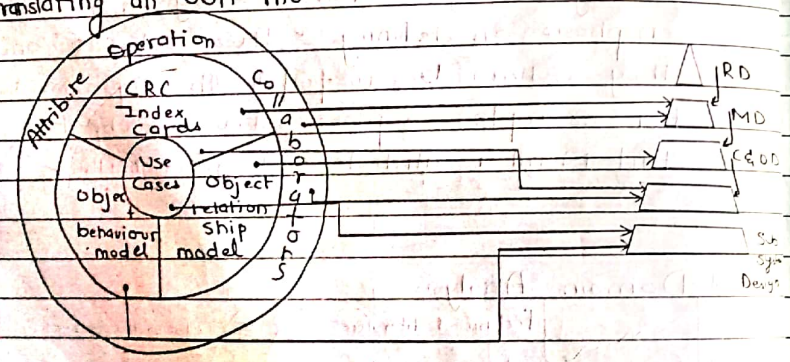
→ Contains the design details that enable each object to communicate with its collaborators.

→ Establishes the external and internal interfaces for the system.

Responsibilities Layer :

→ Contains the data structure and algorithmic design for all attributes and operation for each object.

Important # Translating an OOA model into OOD model :

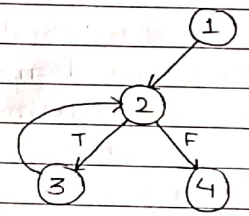


The Analysis Model

Qn. 2 Find the cyclomatic complexity of :

→
```
int a=1, b=1, n, c;          — ①
for (i=1; i<n-2; i++)        — ②
{
    c=a+b;
    a=b;                     — ③
    b=c;
    printf("%d", c);
}                            — ④
```

Step 1 : Data Flow



Step 2 : Cyclomatic Complexity

~~Path 1~~ : V(G) = 2 regions

V(G) = Predicate nodes + 1 = 1 + 1 = 2

V(G) = 4 edges − 4 nodes + 2 = 2

Step 3 : Independent Path

Path 1 : 1 − 2 − 4

Path 2 : 1 − 2 − 3 − 4

Step 4 : Test Cases

Path 1 : 1 − 2 − 4

i =

Path 2 : 1 − 2 − 3 − 4

Qn. 1. Assume that you are building library management system

From above scenario :

1. Draw class diagram
2. Object diagram
3. Use case diagram
4. CRC card
5. ER Diagram (database)

Answer;

1)

Librarian	Student
L_id	S_name
L_name	S_id
L_dog	S_dept
issuebook()	borrow_book()
check_id()	return_book()
grant_permission()	
updated_db()	
add_book()	

Design Issues

(I) Decomposability → System arch sub

(II) Composability → एक ओर designer program लिखे use करने यदि

(III) Understandability → अन्य reference बिना इसे समझ सकें।

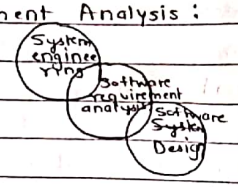
(IV) Continuity → the ability to make small changes in a program and have these change

(V) Protection :- ensure protection if errs occurs in a given module

→ have these change themselve with correspondency changes in just one or a very few module

Chapter : 6 Analysis Concepts and Principles

Requirement Analysis :



- Requirement analysis is a software engineering task that bridges the gap between system level requirements engineering and software design.
- Requirement analysis provides the software designer with a representation of information, function and behaviour that can be translated to data, architectural, interface and component-level designs.

Requirement elicitation

→ Software requirements analysis may be divided into five areas of effort:

1. Problem recognition
2. Evaluation and solution synthesis
3. Modelling (helps to understand functionality of a system)
4. Specification (complete description of behaviour system)
5. Review (objective meet or not)

Requirements Elicitation For Software

- It is the practise of Obtaining the requirements of a system from users, customers and other stakeholders.
- The practise is also sometimes referred to as requirements gathering.

Process of R.E

- Initiating the process
- Facilitated Application Specification Technique (FAST)
- Quality Function Deployment
- Use - Cases

1 # Initiating the process

The most commonly used requirements elicitation technique is to conduct a meeting or interview.

- The analyst starts by asking context-free questions.
- The first set of context-free question focuses on the customer, the overall goals, and benefits
- The next set of questions enables the analyst to gain a better understanding of the problem and the customer to voice his or her perceptions about a solution.
- The final set of questions focuses on the effectiveness of the meeting.

2 # FAST

- This approach encourages the creation of a join team of customers and developers.
- They work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements.

3 # Quality Function Deployment

- QFD is a quality management technique that translates the needs of the customer into technical requirements for software
- QFD "concentrates on maximizing customer satisfaction from the software engineering process".
- QFD identifies three types of requirements :
  - → Normal requirements
  - → Expected "
  - → Exciting "

4 # Use-Cases

- Provide a description of how the system will be used.
- The use-case describes the manner in which an actor ~~action~~ interacts with the system.

Analysis principles

→ All analysis methods are related by a set of operational principles

1. The information domain of a problem must be represented and understood.
2. The functions that the software is to perform must be defined. (as a consequence of events)
3. The behaviour of the software must be represented

In addition, Davis :

1. Understand the problem before you begin to create the analysis model.
2. Develop prototypes that enable a user to understand how human/machine interaction will occur.
3. Record the origin of and the reason for every requirement
4. Use multiple views of requirements
5. Rank requirements
6. Work to eliminate ambiguity

Information Domain

- All software applications can be collectively called data processing.
- Data (numbers, text, images, sounds, video, etc) and control (events) both reside within the information domain of a problem.
- It contains 3 different views of the data and control as each is processed by a computer prog
  1. Information content and relationships (the data model)
  2. Information flow, and
  3. Information Structure

1. Information content represents the individual data and control objects that consitute some larger collection of information transformed by the software.
2. Info flow represents the manner in which data and control change as each move through a system.
3. Info structure represents the internal organization of various data and control items. Data Structure refers to the design and implementation of information structure within the software.

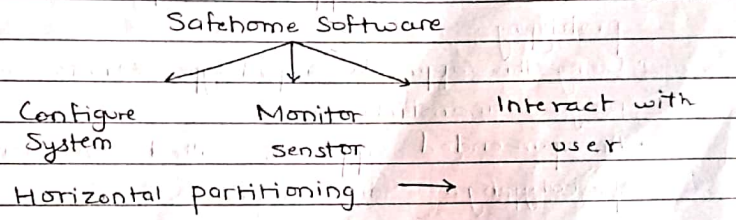
Modelling

1. Functional models:
   - When functional models of an application are created, the software engineer focuses on problem specific functions.
   - The functional model begins with a single context level model (i.e the name of the software to be built).
   - over a series of iterations, more and more functional detail is provided, until a thorough delineation of all system functionality is represented.
2. Behaviour Model
   - Most software responds to events from outside wor
   - The response characterstics forms the basis of the behavioral model.

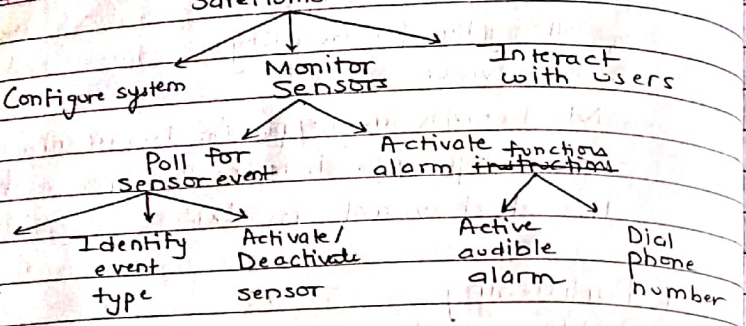
Partitioning

- Partitioning decomposes a problem into its consituent par
- Concenptually, we establish a hierarchical representation of function or information and then partition the uppermost element by
  1. exposing increasing detail by moving vertically in the hierarchy or
  2. Functionally decomposing the problem by moving horizontally in the hierarchy.
- Considering the example of "safe home system" it can be further partitioned as horizontal and vertical

Horizontal



Vertical Partitioning



Essential and Implementation View

- An essential view of software requirements presents the functions to be accomplished and information to be processed without regard to implementation detail.
- The implementation view of software requirements presents the real word manifestation of processing function and information structures.

Prototype

→ Selecting the prototyping approach.

- The close ended approach is often called throwaway prototyping.
- Using this approach a prototype servers solely as rough demonstration.
- An open-ended approach called evolutionary prototyping.

Software Requirements Specification (SRS)

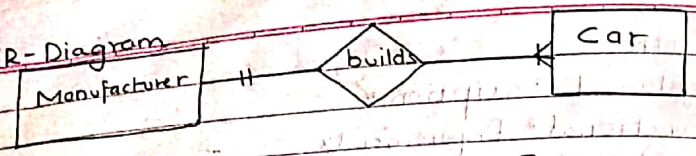
- A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform.
- It is usually signed off at the end of requirements engineering phase.

1. Intro
2. System Description
3. Functional Requirements
   - i) System Features
     - System feature1
     - System feature2
   - ii) Use cases
     - Use case diagram
     - Use case1
     - Use case2
   - iii) Entity Relationship Diagram
   - iv) Data Dictionary
     - Entity 1
     - Entity 2
4. External Interface Requirements
5. Technical Requirements (non-functional)
6. open Issues

Specification Review

- A review of the software requirements specification (and/or prototype) is conducted by both the software developer and the customer.
- Because the specification forms the foundation of the development phase, extreme care should be taken in conducting the review.
- The review is first conducted at a macroscopic level.
- Once the review is complete, the software requirement specification is 'signed off' by both the customer and the developer.
- The specification becomes a 'contract' for software development.

ER-Diagram



Function modeling and Information Flow

- Information is transformed as it flows through a computer based system.
- The system accepts input in a variety of forms; applies hardware, software and human elements to transform it; and produces output in a variety of forms.

vimp # Data Flow Diagrams

- A data flow diagram is a graphical representation that depicts information flow and the transforms that are applied as data move from input to output.

Representation of Data Flow

External Entity

Process

Data Store

Data Flow →

- A rectangle is used to represent an external entity.
- A circle (sometimes bubble) represents a process or transform that is applied to data (or control) and changes it in some way.
- An arrow represents one or more data items (data objects).
- The double line represents a data store - stored information that is used by the software.

System

Req Specify { SSA → ERD flow, DFD
OOA UML

Struct / Static Dynamic / Behavior

DFD levels:

- 0 - level DFD :
- 1 - level DFD
- 2 - level DFD

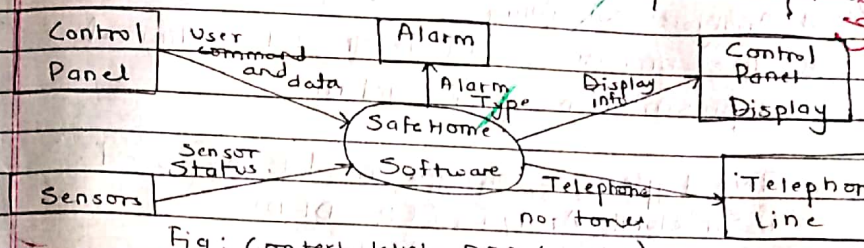


Fig: Context level DFD (level 0) for SafeHome

The Data Dictionary

- Name
  → The primary name of the data or control item, the data store or an external entity.
- Alias
  → Other names used for the first entity.
- Where-used / how-used
  → a listing of the processes that use the data or control item and how it is used (eg. input to the process, output from the process, as a store, as an external entity).
- Content description
  → a notation for representing content.

- Supplementary information
→ other information about data types, preset values (if known), restrictions or limitations and so forth

Analysis Modelling
- The analysis model is the 1st technical representation of a system.

* Two types of analysis model are used:
  i) Structural Analysis (ER, DFD)
  ii) Object Oriented Analysis

Steps of Analysis Modelling
1. Data, functional and behaviour, are modeled using a number of different diagram formats.
2. Data modeling defines data objects, attributes and relationship.
3. Functional model indicate how data are transformed within a system.
4. Behavioural Modelling depict the impact of events.
5. Once preliminary models are created, they are refined and analyse to access their clarity, and consistency. A specification incorporating the model is created and then validated by both software engineers and users.

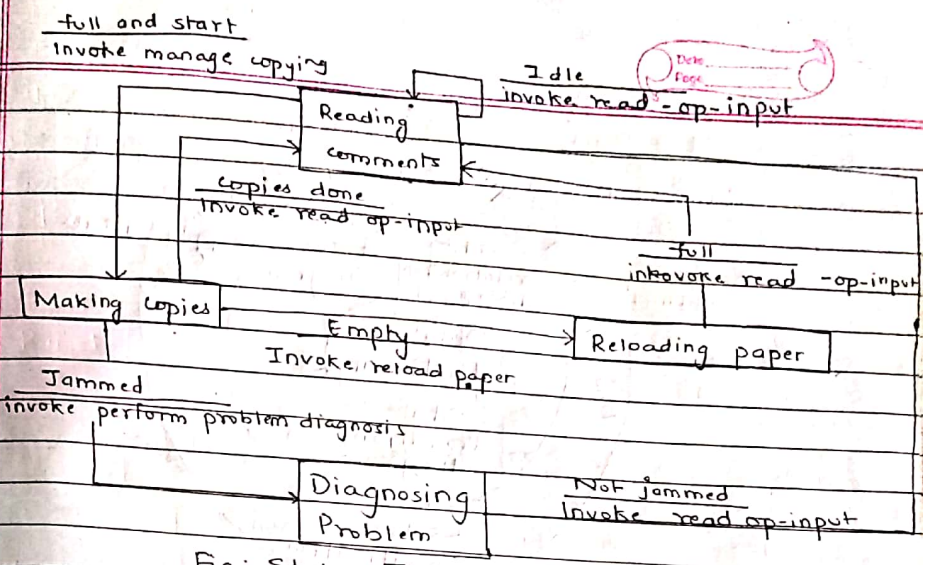


Fig: State Transition Diagram

The elements of the Analysis model
→ The analysis model must achieve three primary objectives:
1. To describe what the customer requires.
2. To establish a basis for the creation of a software design and
3. To define a set of requirements that can be validated once the software is built.

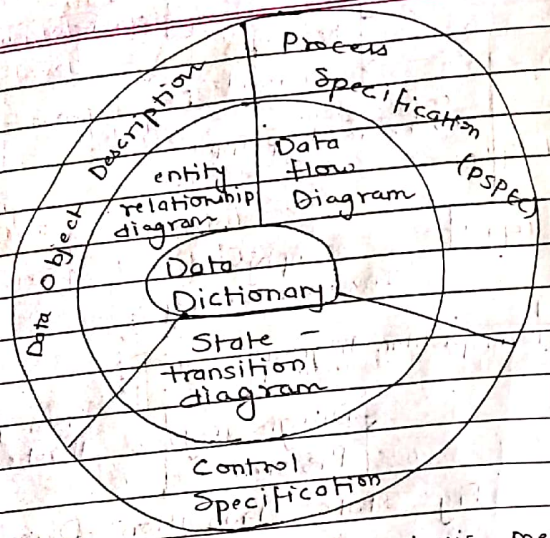
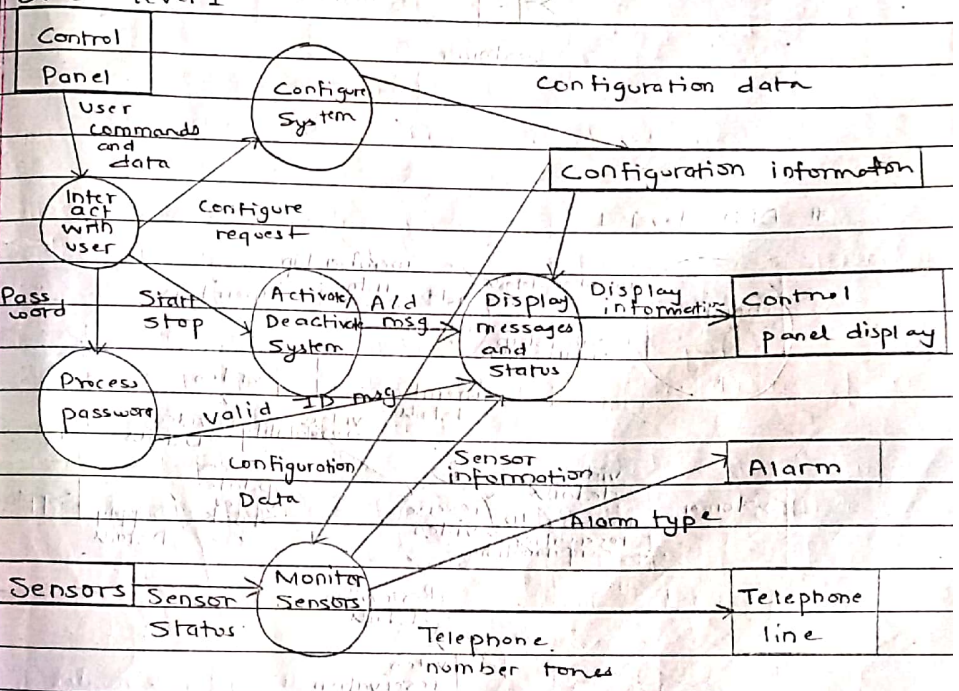


Fig: The structure of the analysis method.

Data Flow Diagram (DFD)

- The DFD maybe used to represent a system or software at any level of abstraction.
- DFDs may be partitioned into levels that represent increasing information flow and functional detail.
- Therefore, the DFD provides a mechanism for functional modelling as well as information flow modelling.
- A level 0 DFD, also called a fundamental system model or a context model, represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively.
- Each of the processes represented at level 1 is a sub function of the overall system depicted in the context model.
- Each of the bubbles maybe refined or layered to depict more details.

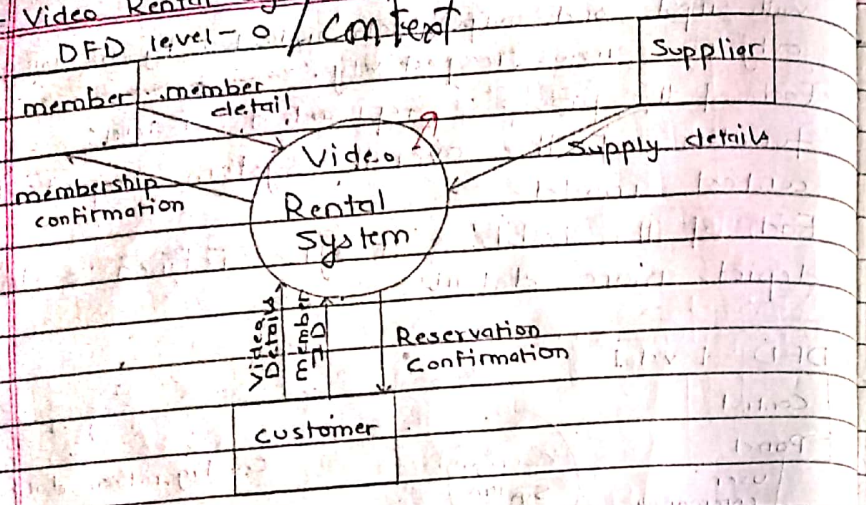
DFD level 1



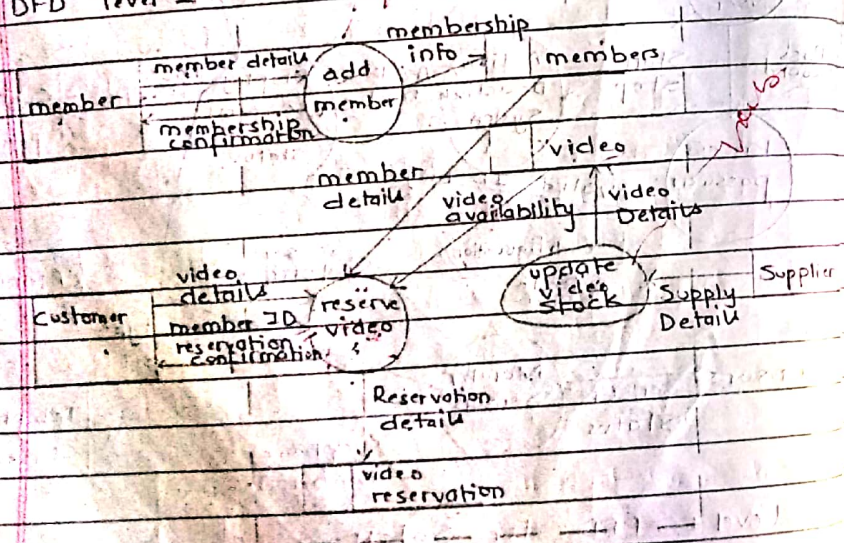
Level 1 DFD for SafeHome

Video Rental System — Context Diagram

DFD level-0 / Context

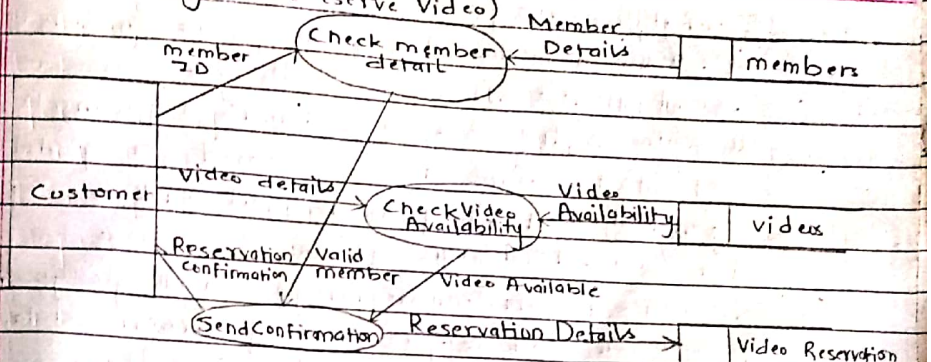


DFD level 1



{PU 2016 4th Spring}

Level 2 Diagram (Reserve Video)



→ member.
→ video availability

Functional primitive

Progress

Data Dictionary.

Example: The data dictionary provides us with a precise definition of telephone number for the DFD in question. The data dictionary entry begins as follows:

→

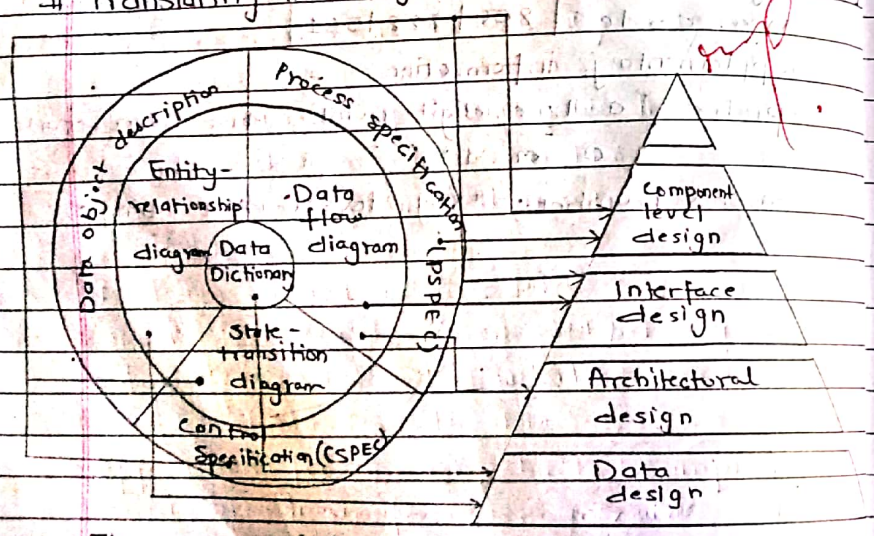
- Name: telephone number
- Aliases: none (Alias)
- Where used/how used: assess against set-up (output)
  dial phone (input)
- Description: tel number = [local number/long distance number]
- local number = prefix + access number
- long distance number = 1 + area code + local number
- area code = [800 | 888 | 561]
- Supplementary information
  - prefix = * a three digit number that never starts with 0 or 1 *
  - access number = * any four number string *

Chapter 7: Design Concept & Principles

Design Concept

- Design is a meaningful engineering representation of something that is to be built.
- It can be traced to a customer's requirements and at the same time assessed for quality against a set of pre-defined criteria for 'good' design.
- In the software engineering context, design focuses on four major areas of concern: data, architecture, interfaces and components.

Translating the analysis model into a software design



The analysis model — The design model

- The data design transforms the information domain model created during analysis into the data structures that will be required to implement the software.
- The data objects and relationships defined in the entity relationship diagram and the data dictionary provide the basis for the data design activity.

- The architectural design defines the relationship betn major structural elements of the software, the 'design patterns' that can be used to achieve the requirements that have been defined for the system, and the constraints that affect the way in which architectural design patterns can be applied.

- The interface design describes how the software communicates within itself, with systems that interoperate with it, and with humans who use it.
- An interface implies a flow of information (eg: data and/or control) and specific type of behaviour.
- Therefore, data and control flow diagrams provide much of the information required for interface design.

- The component-level-design transforms structural elements of the software architecture into a procedural description of software components.
- Info obtained from the PSPEC, CSPEC, and STD serves as the basis for component design.

- During design we make decisions that will ultimately affect the success of software construction and, as important, the ease with which software can be maintained.

Why is design so important?

- The importance of software design can be stated with a single word – quality.

Imp # Design Guidelines

1. A design should exhibit an architectural structure that
   (a) has been created using recognizable design patterns.
   (b) is composed of components that exhibit good design characteristics;
   (c) can be implemented in an evolutionary fashion.
2. A design should be modular;
3. A design should contain distinct representations of data, architecture, interfaces, and components.
4. A design should lead to data structures that are appropriate for the objects to be implemented and are drawn from recognizable data patterns.
5. A design should lead to components that exhibit independent functional characteristics.
6. A design should lead to interfaces that reduce the complexity of connections betn modules and with the external environment.
7. A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.

Design Principles

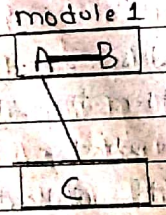
- Software design is both a process and a model.
- The design process is a sequence of steps that enable the designer to describe all aspects of the software to be built.
- The design model is the equivalent of an architect's plans for a house.
- It begins by representing the totality of the thing to be built and slowly refines the thing to provide guidance for constructing each detail.
- Davis suggests a set of principles
  1. The design process shouldn't suffer from 'tunnel vision'.
  2. The design should not reinvent the wheel.
  3. The design should 'minimize the intellectual distance' betn the software and the problem as it exists in the real world.
  4. The design should exhibit uniformity and integration.
  5. The design should be structured to accomodate change.
  6. The design should be structured to degrade gently, even when aberrant data, events or operating conditions are encountered.

7. Design is not coding, coding is not design.
8. The design should be reviewed to minimize conceptual (semantic) errors.

Coupling and Cohension (Imp for short notes)

- Coupling is also the indication of the relationship betn modules. It is concept of Inter-module. Coupling has also many types but usually low coupling is for software.

- Cohesion is the indication of the relationship within the module. It is concept of intra-module. Cohension has many types but usually high cohesion is good for software.



Design Concepts

- Design concepts provide the foundation from which more sophisticated sign methods can be applied.
- Each helps the software engineer to answer the following questions.
  - What criteria can be used to partitioned software with individual components?
  - How function or data is structure detail separated from a conceptual representation of the software?
  - What uniform criteria define the technical quality of a software design?

1. Abstraction

→ Procedural Abstraction ~~ett~~ is a named sequence of instructions that has a specific and limited function.

→ Data Abstraction is a named collection of data that describes a data object.

→ Control Abstraction is the third form of abstraction used in software design; control abstraction implies a program control mechanism without specifying internal details.

2. Refinement

→ A program is developed by successively ~~refing~~ refining levels of procedural details.

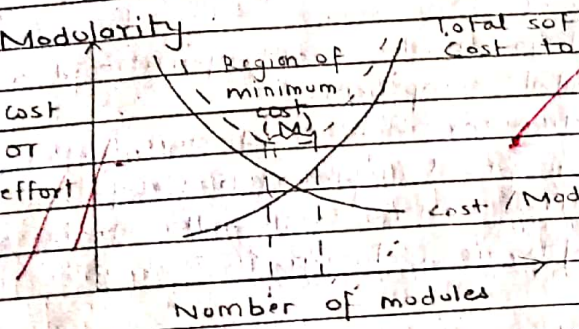
→ Refinement is actually a process of elaboration.

→ Refinement causes the designer to elaborate on the

original statement, providing more and more details as each successive refinement occurs.

→ Abstraction and refinement are complementary concepts

3. Modularity



→ Modularity is the most common manifestation of 'separation of corners'.

→ Software is divided into separately named and addressable components, sometimes called modules that are integrated to satisfy problem requirements.

→ If you submitted software indefinitely the effort required to develop it will become neglibly small

→ the effort (cost) to develop an individual software module does decrease as the total number of modules increases.

→ However, as the no. of modules grows, the effort (cost) associated with integrating the modules also grow.

→ These characteristics lead to ~~or~~ a total cost or effort curve show in the above figure.

4. Software Architecture

→ It refers to the overall structure of the software and the ways in which that software provides conceptual integrity for a system.

→ Architecture is the hierarchical structure of program components (modules), the manner in which the components interact and the structure of data that are used by the components.

→ A set of properties that should be specified as part of an architectural design.

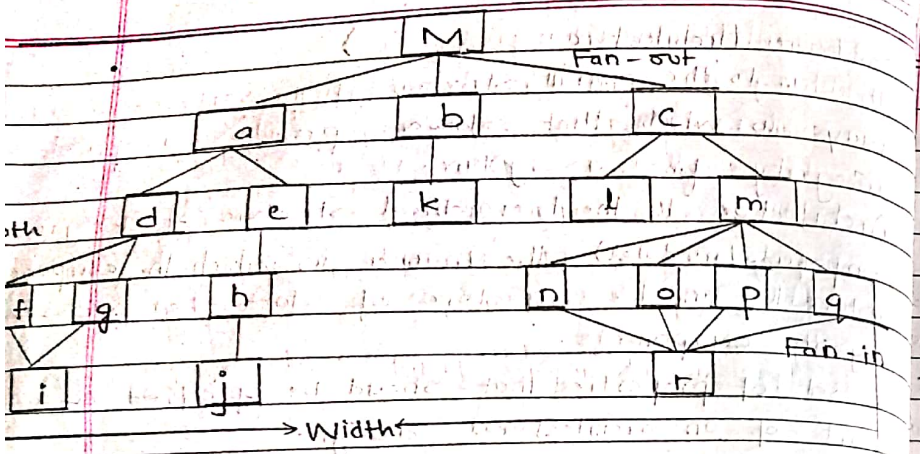
1. Structural properties (components of system modules, objects, filters)
2. extra-functional properties (performance, capacity, reliability)
3. Families of related systems.

Control Hierarchy

- Control Hierarchy, also called program structure, represents the organisation of program concepts (modules) and implies a hierarchy of control.

It doesn't represent procedural aspects of software such as sequence of processes.

Different notations are used to represent control hierarchy for those architectural styles that are amenable to this representation.



- Fan-out is a measure of the number of modules that are directly controlled by another model.
- Fan-in indicates how many modules directly a given module

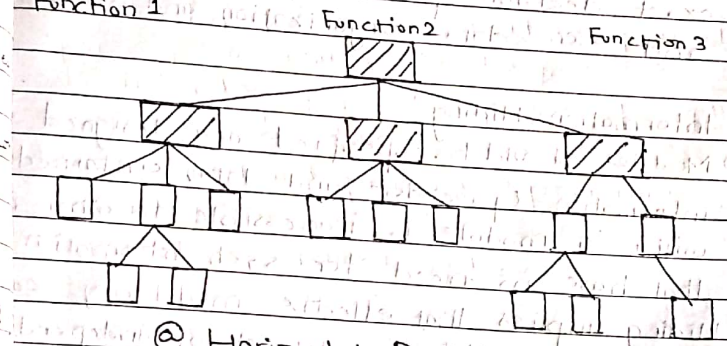
6. Structural Partitioning

- If the architectural style of a system is hierarchical program structure can be partitioned both horizontally and vertically.
- Horizontal partitioning defines separate branches the modular hierarchy for each program function.
- Control modules, (represented in a darker shade) are used to coordinate communication betn and execution of the functions.
- The simplest approach to horizontal partitioning three - partitions - input, data transformation (called processing) and output.

- Horizontal Partitioning

→ Partitioning the architecture horizontally provides a number of distinct benefits such as:

1. Software that is easier to test.
2. Software that is easier to maintain.
3. Propagation of fewer side effects.
4. Software that is easier to extend.



(a) Horizontal Partitioning.

Data Structure

S is a representation of the logical relationship among the individ lements of data.

he structure information will invariably affect the final rocedure design, data structure is as important as rogram structure to the representation of software architect S dictates the organization methods of access, degree associativity, and processing alternatives for information scalar item is the simplest of all data structures

8. Software Procedure

→ Program structure defines control hierarchy with regards to the sequence of processing and d.

→ Software Procedure focuses on the processi details of each module individually.

→ Procedure must provide a precise specifica of processing, including sequence of events, exact decision points, repetitive operations and even data organization and structure

9. Information Hiding

→ Modules should be specified and designed so that information (procedure and data) contained within a module is inaccessible to other modules that have no need for such information

→ Hiding implies that effective modularity can achieved by defining a set of independen modules that communicate with one anoth only that information necessary to achiev software function.

F. Software Architecture.

• The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

Importance of Software Architecture.

1. Representations of software architecture enable communicati betn all parties (stakeholders) interested in the development of a computer-based system.
2. The architecture highlights early design decisions.
3. Architecture "constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together".

Data Design V. Imp

• Data design creates a model of data and/or informat that is represented at a high level of abstraction (the customer/ user's view of data).

• Data design plays an important role in following cases
- At the program component level.
- At the application data design
- At the business level.

Data Modeling, Data Structures, Databases and the Data Warehouse.

- The data objects defined during software requirements analysis are modeled using entity / relationship diagrams and the data dictionary.
- Characteristics that differentiate a data warehouse from the typical database:
  1. Subject Orientation
  2. Integration
  3. Time Variance
  4. Non volatility

Data Design at the Component Level

- Data Design at the component level focuses on the representation of data structures that are directly accessed by one or more software components.
  1. The systematic analysis principles applied to function and behaviour should also be applied to data.
  2. All data structures and operations to be performed on each should be identified.
  3. A data dictionary should be established and used to define both data and program design.

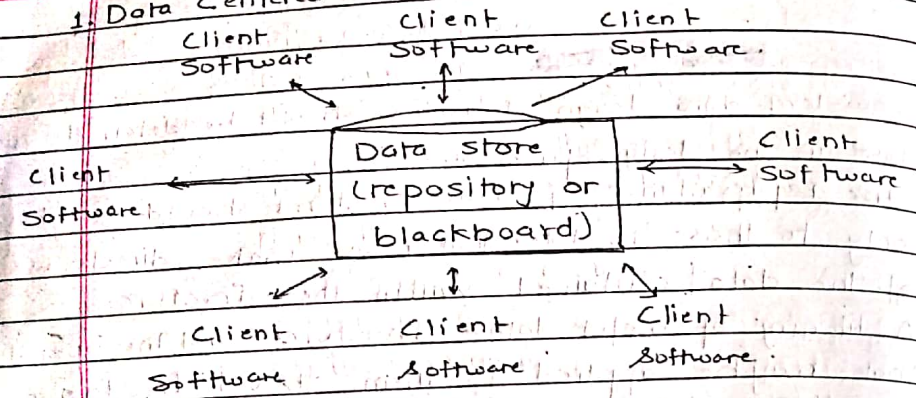
Principles for data specification

4. Low-level data design decisions should be deferred until late in the design process.
5. The representation of data structures should be known only to those modules that must make direct use of the data contained within the structure.
6. A library of useful data structures and the operations that maybe applied to them should be developed.
7. A software design and programming language should support the specification and realization of abstract data types.

Architectural Styles

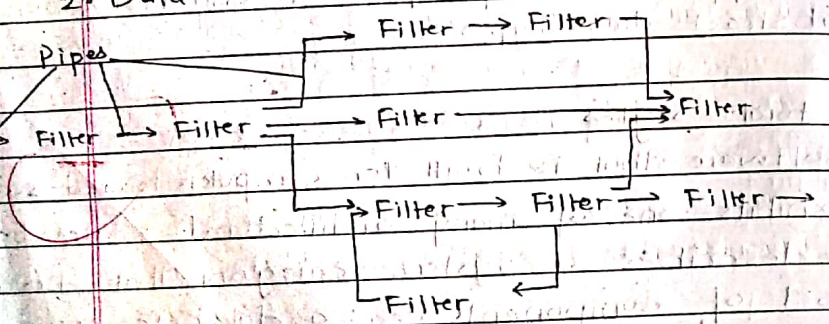
- The software that is built for computer-based systems also exhibits one of many architectural styles. Each style describes a system category that encompasses
  1. A set of components (eg: a database, computing modules) that perform a function required by a system.
  2. A set of connectors that enable "communication, co-ordinations and cooperation" among components.
  3. Constraints that define how components can be integrated to form the system; and
  4. Semantic models that enables a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

1. Data Centered Architecture :

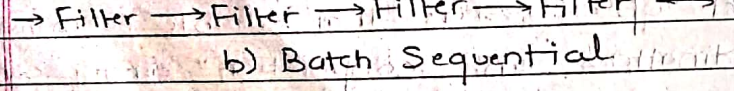


→ promotes integrability

2. Data-flow architectures :



a) Pipes and Filters

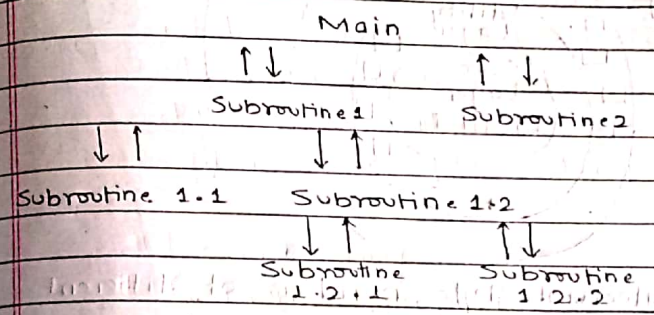


b) Batch Sequential

3. Call and return architectures

- Main program / subprogram architectures
- Remote procedure call architectures

Remote Procedure Calls (RPC)

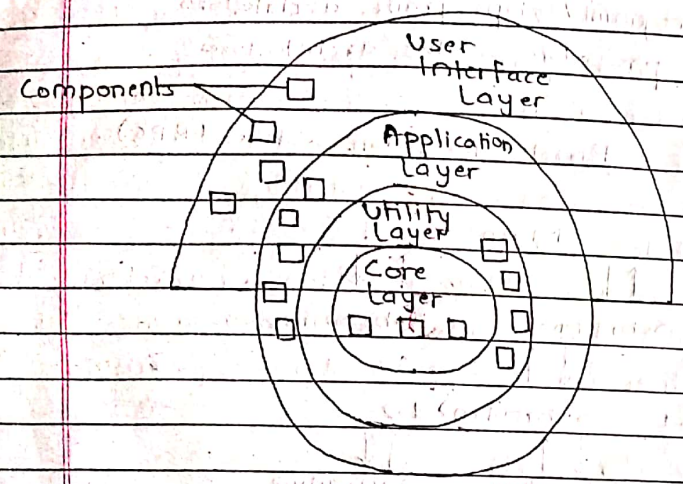


- The components of a main program / subprogram architecture are distbtd across multiple computers on a network.

4. Object-oriented architectures

- The components of a system encapsulate data and the operations that must be applied to manipulate the data.
- Communication and coordination between components is accomplished via message passing.

5. Layered Architecture



- In this architectural style, numbers of different layers are defined, each accomplishing operation that progressively become closer to the machine instruction set. At the outer layer, components service user interface operations.

Mapping Requirements into a Software Architecture

1. The type of information flow is established;
2. Flow boundaries are indicated;
3. The DFD is mapped into program structure;
4. Control hierarchy is defined
5. Resultant structure is refined using design measures and heuristics; and
6. The architectural description is refined and elaborated.

Transform Flow

- External data must be converted into an internal form of processing.
- Information enters the system along paths that transform external data into an internal form.
- These paths are identified as incoming flows.
- At the kernel of the software, a transition occurs.
- Information flow is often characterized by a single data item, called a transaction that triggers other data flow along of many paths.

Level Factoring

Transform Mapping

OOA what | OOD how

Abstr

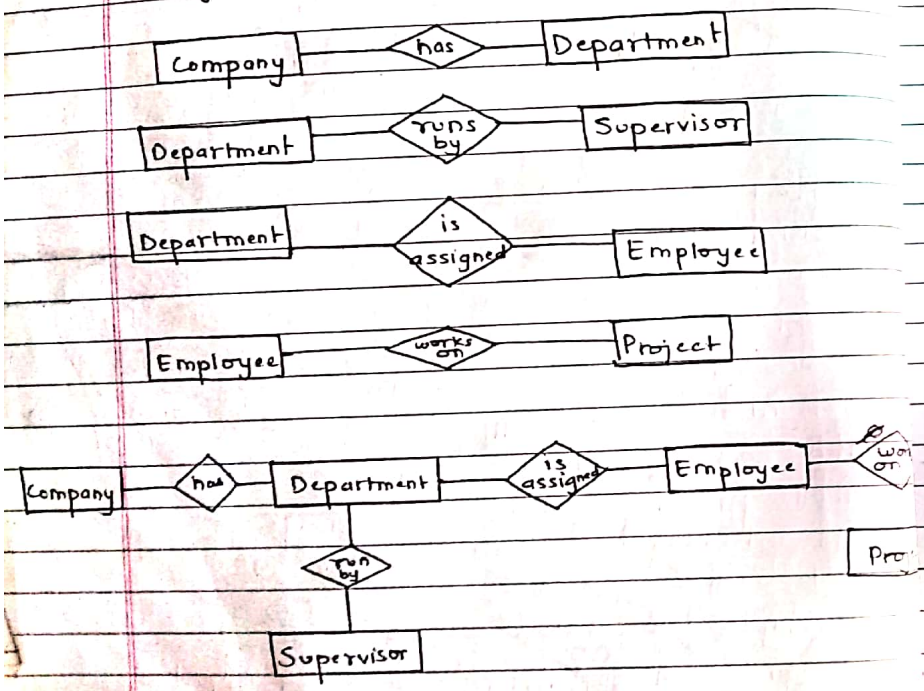
Comm

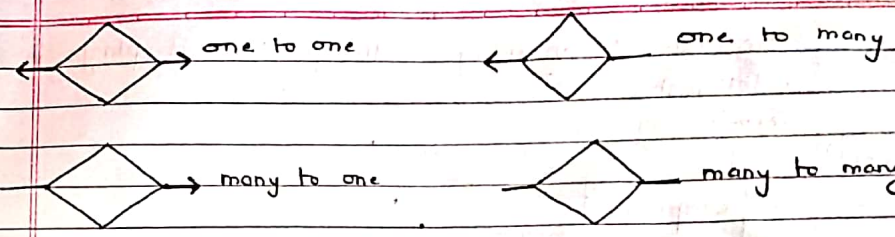
Ans; 1. identify entities :- company, departments, supervisor, Employees, Project

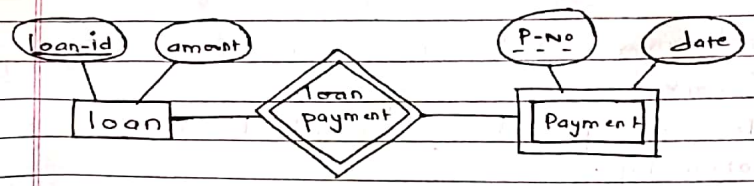
2. find relationship :-

	Company	Department	Supervisor	Employee	Proj
Company		has			
Dept	assigned to		run by	is assigned	
Supervisor		runs			work
Employee		belongs to		uses	
Project					

3. rough ERD





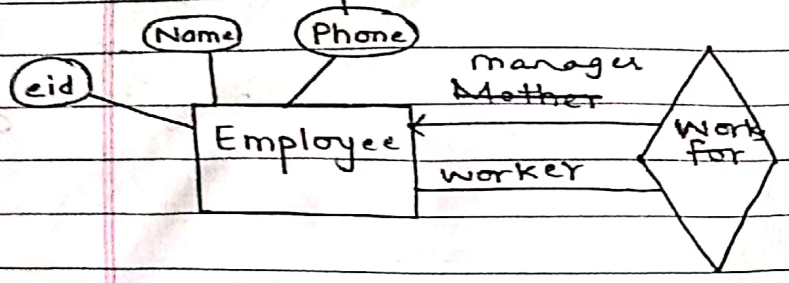


How to develop a ERD?

1. Identify entities
2. Find relationship
3. Draw rough ERD
4. Fill in cardinality
5. Define primary keys
6. Draw key based ERD
7. Identify attributes
8. Map attributes
9. Draw fully attributed ERD
10. Check final ERD

Eg: A company has several departments. Each dept has a supervisor and atleast one employee. Employees must be assigned to atleast one but possibly more depts. Atleast one employee is assigned to a project but an employee maybe on vacation and not assigned to any projects. The important data fields are names of depts, projects, supervisors and employees, as well as the supervisor and employee no. and a unique project no.

The role of an entity is the function it plays in a relationship.



- Here 'manager and worker' are roles of entity employee in 'works for' relationship.
- Roles are optional and clarified semantics of a relationship.

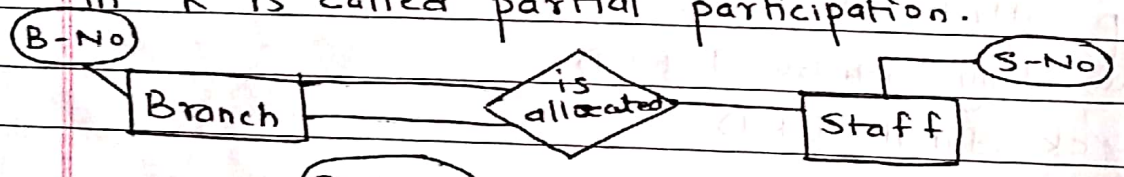
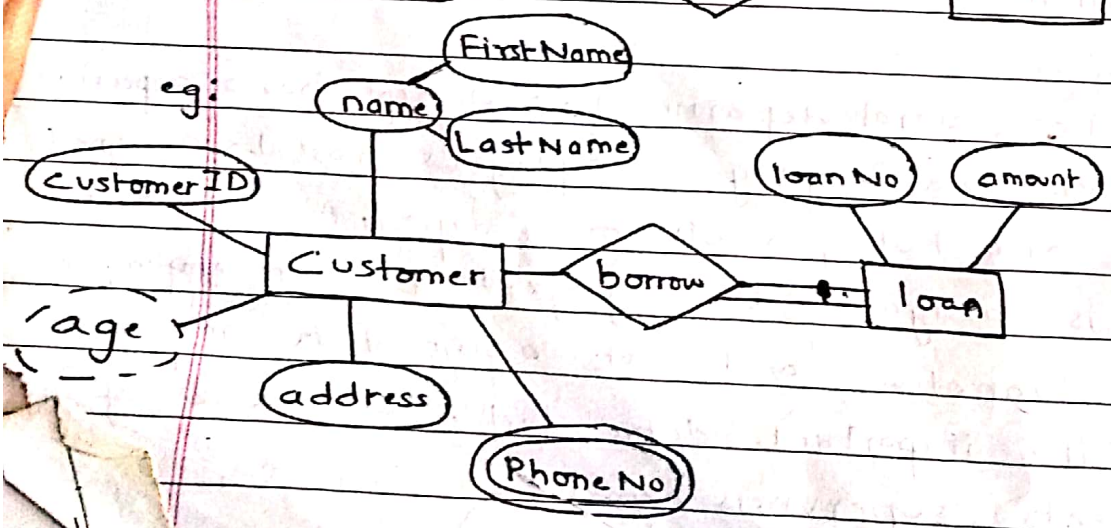
Participation Constraint

1. Total participation :

If every entity in entity set 'E' participates in atleast one relationship in 'R' is called total participation.

2. Partial participation :

If only some entities in 'E' participates in relationship in 'R' is called partial participation.



eg: