4.1 Software Design :-

Definition :-

" Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation".

S/w design is the first step in SDLC(S/w development life cycle), which moves the concentration from problem domain to Solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

Ref. :- R4

4.2 Conceptual & Technical Designs :-

→ To transform req's into a working system, designers must satisfy both customers & the system (or s/w) builders.

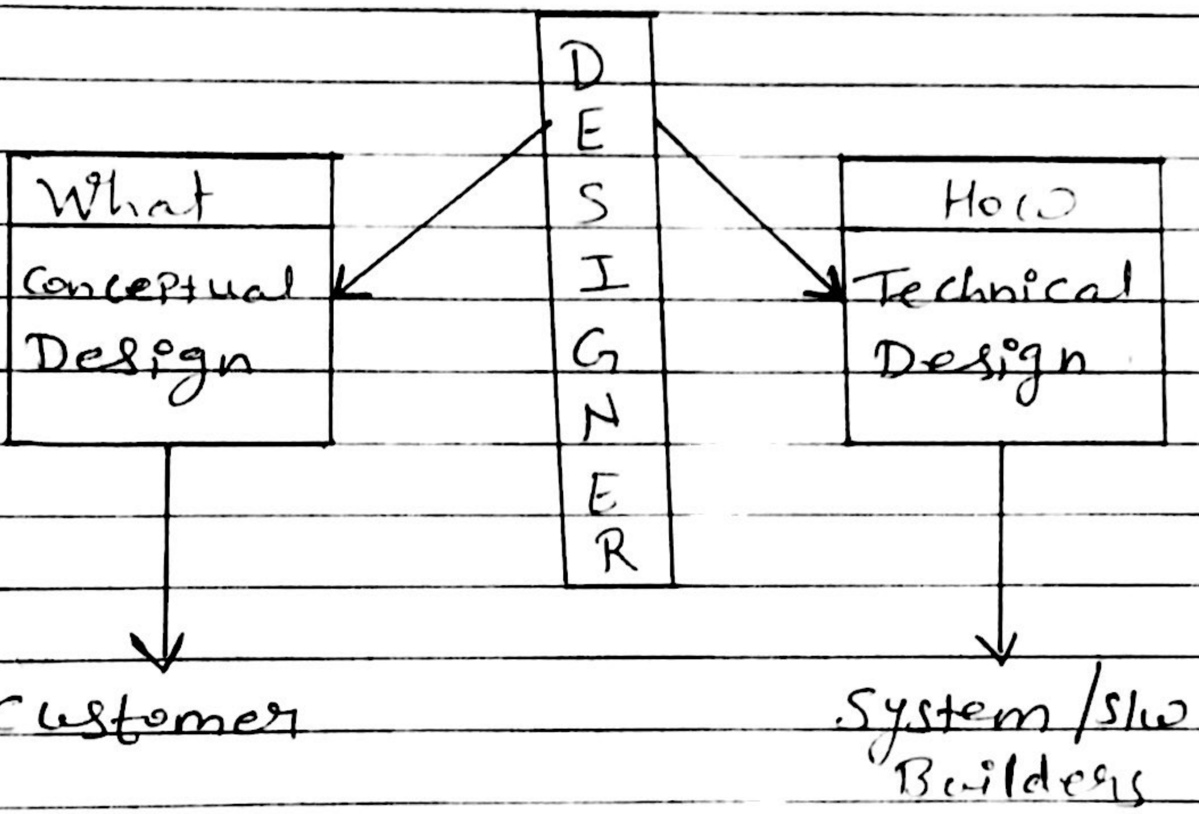
→ Customers should understand What the system is to do.

→ System builders must understand how to do.

→ To accomplish these, design is divided into two parts.

1) Conceptual Design.

2) Technical Design.



Conceptual & Technical Design

1) **Conceptual Design:-**

→ Written in customer's language.

* Conceptual designs also answers:

→ Where will the data come from?

→ What will happen to data in the system?

→ How will the system look to users?

→ What choices will be offered to users?

→ What is the timing of events?

→ How will reports & screens look like?

* Conceptual designs describes following.

technical terminology

→ It contains no technical Jargon (if it does, define it

→ It describes the functions of the system.

→ It is independent of implementation.

→ It is linked to the req's. documents.

2) Technical Design:-

* Technical design describes:

→ Hardware configuration.

→ Software needs

Name of Lecturer Abhishek Jain.

→ I/O of the system (Interfaces).

→ Network Architecture.

→ Data structures & data flow.

Ref. :- R5, R6

4.3 Characteristics of Good Design:-

Good S/W design should exhibit:

a quality of being steady

1) **Firmness :-** A program should not have any bugs after implementation.

article of commerce

2) **Commodity :-** A program should be suitable for the purposes for which it was intended. (business logic).

3) **Delight :-** The experience of using the program should be pleasurable one.

4) **Reusability :-** Designing the system so that you can use pieces of it in other systems.

5) Portability :- It must be portable enough to move the system to another environment.

6) Ease of Maintenance :-

Design of the system should be self explanatory. If any error or bug occur, then it is easy to maintain or fix.

7) Extensibility :- You can change a piece of the system without affecting other pieces.

Ref :- R7

4.4 Design Principles :-

S/w design can be viewed as both a process & a model.

"The design process is a sequence of steps that enable the designer to describe all aspects of the S/w to be built. However, it is not merely a cookbook: for a competent & successful design, the designer must use creative skill, past experience, a sense of what makes "good" software, & have a commitment to quality.

* Davis suggests a set of principles for s/w design.

1) The design Process Should not suffer from "tunnel vision".

A good designer Should consider alternative approaches, Judging each based on the req's of the problem, the resources available to do the Job.

2) The design Should be traceable to the analysis model.

Because a Single element of the design model often traces to multiple req's, it is necessary to have a means for tracking how req's have been Satisfied by the design model.

3) The design Should not reinvent the Wheel.

Systems are constructed using a set of design patterns, many of Which have likely been encountered before. These patterns Should always be chosen as an alternative to reinvention

Time is short & resources are limited. Design time should be invested in representing truly new idas & integrating those patterns that already exists.

4) The design should "minimize the intellectual distance" b/w the s/w & the problem as it exists in the real world.

That is, the structure of the s/w design should (whenever possible) mimic the structure of the problem domain.

5) The design should exhibit uniformity and integration;

A design is uniform if it appears that one person developed the entire thing. Rules of style & format should be defined for a design team before design work begins. A design is integrated if care is taken in defining interfaces b/w design components

6) The design should be structured to accomodate change.

The design concepts discussed in the next section enable a design to achieve this principle.

Name of Lecturer Abhishek Jain

7) The design should be structured to degrade gently, even when aberrant (abnormal) data, events or operating conditions are encountered

Well designed s/w should never "bomb". It should be designed to accommodate unusual circumstances, & if it must terminate processing, do so in a gracefull manner.

8) Design is not coding, coding is not design.

Even when detailed designs are created for program components, the level of abstraction of the design model is higher than source code. The only design decisions made at the coding level address the small implementation details that enable the procedural design to be coded.

9) The design should be assessed for quality as it is being created, not after the fact.

A variety of design concepts & design measures are available to assist the designer in assessing quality.

10) The design should be reviewed to minimize conceptual errors :-

A design team should ensure that major conceptual elements of the design (omissions, ambiguity, inconsistency) have been addressed before worrying about the syntax of the design model.

Ref. :- R8

4.5 Design Guidelines :-

The following guidelines create a base to develop a S/w design

1) Review the req's. speifications & gather the functional charactistics of the system.

2) Review & expand the external interfaces, user dialogues & report formats developed during req's. analysis.

3) Review & refine the data models (ERD, CFD, DFD..)

Name of Lecturer Abhishek Jain

developed during req's analysis

4) Identify functional & data abstraction

5) Define the modularization criteria to be used in establishing system structure.

6) Apply the techniques of your particular design method to establish system structure.

7) Iterate steps 1 through 7 untill the suitable structure is achieved.

8) Verify that the resulting system structure satisfies the req's.

9) Conduct the preliminary design review

10) Conduct the critical design review

11) Redesign as necessary

Ref. :- R8

4.6 | **Design Fundamentals :-**

A set of fundamental s/w design concepts has evolved over the past four decades, each providing the s/w designer with a foundation from which more sophisticated design methods can be applied.

Each concept helps the s/w engineer to answer the following questions:

1) What criteria can be used to partition s/w into individual components?

2) How is function or data structure detail separated from a conceptual representation of s/w?

3) Are there uniform criteria that define the technical quality of a software design?

The fundamental design concepts are

1) **Abstraction :-** Allows designers to focus on solving a problem without being concerned about irrelevant lower level details (procedural abstraction, data abstraction)

Name of Lecturer Abhishek Jain

2) **Refinement :-** It is a process of elaboration, where the designer provides successively more detail for each design component.

3) **Modularity :-** Software architecture is divided into components called modules.

4) **Software Architecture :-**

It refers to the overall structure of the S/w & the ways in which that structure provides conceptual integrity for a system.

A good S/w architecture will give good return on investment with respect to the desired outcome of the project, ex; in terms of performance, quality, schedule & cost.

5) **Control Hierarchy :-**

A program structure that represents the organization of a program component & implies a hierarchy of control.

6) Structural Partitioning :-

The program structure can be divided both horizontally & vertically. Horizontally partitioning defines three partitions (input; data transformations; & output). vertical partitioning distributes control in a top-down manner (control decisions in top-level modules & processing work in the lower level modules).

7) Data Structure :-

It is a representation of the logical relationship among individual elements of data.

8) Software Procedure :-

It focuses on the processing of each modules individually.

9) Information Hiding :-

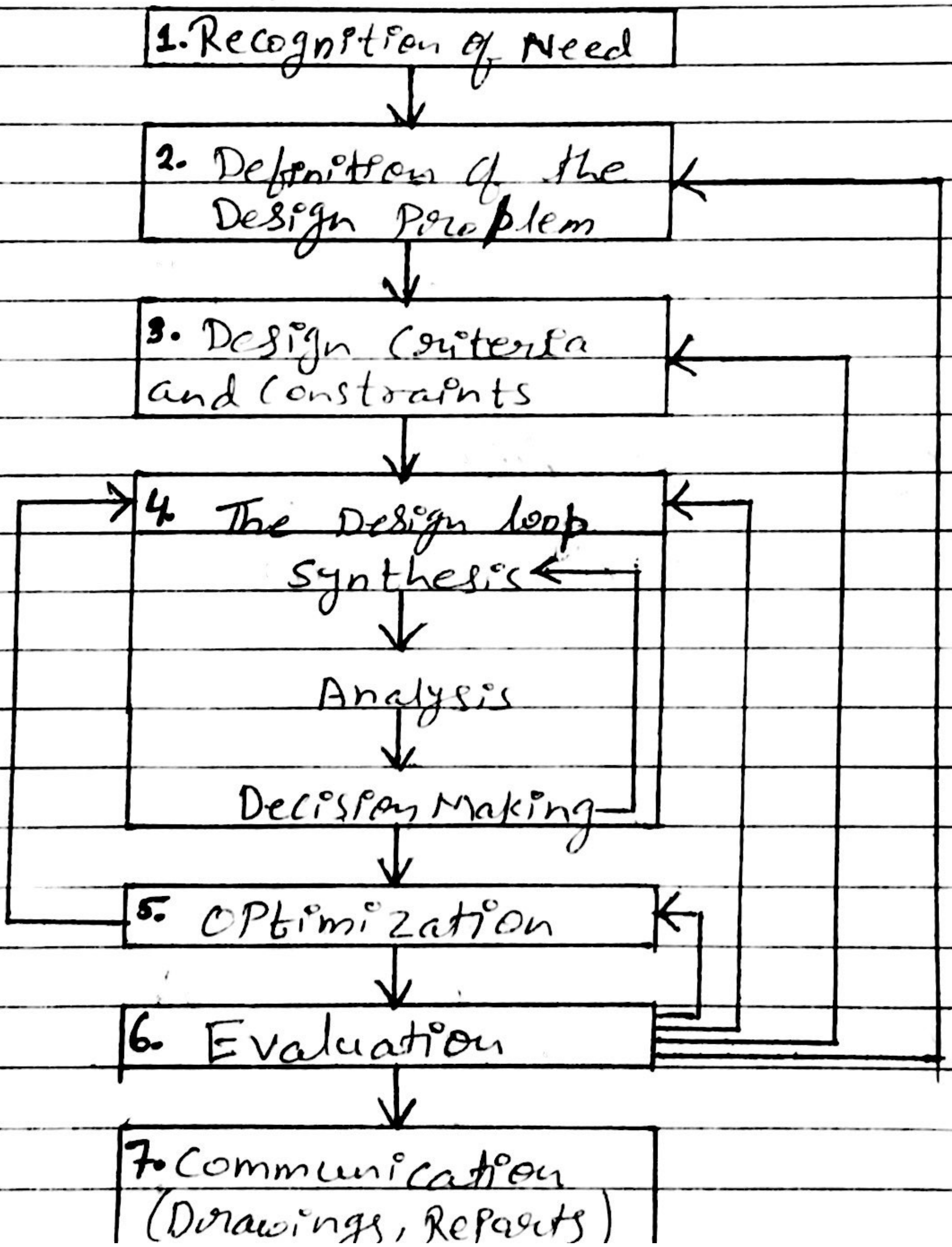
Modules should be specified & designed so that information contained within a module is inaccessible to other modules that have no need for such information.

Ref. :- RB

Name of Lecturer Abhishek Jain

4.7 Software Design Process :-

The s/w design Process basically follows an ordered sequence of steps which must be followed one-by-one to achieve final Product. The below diagram depicts the design Process.


1. Recognition of Need
2. Definition of the Design Problem
3. Design Criteria and Constraints
4. The Design loop
Synthesis
Analysis
Decision Making
5. Optimization
6. Evaluation
7. Communication (Drawings, Reports)

A design for a S/w product Should provide the information about the desired o/p, required environment & economic conditions etc.

Diagram Description.

→ First the appropriate data & information are collected which are related to the problem. why the design is needed is understand.

→ Now, each component of the design was defined. Functionality of the each component are also defined.

→ In the next section the criteria & constraints are defined by which the design is going to developed. criteria is established to have physical realizations constrains refers to rules & models to be followed.

→ The next step of design Process is an iterative loop of design in which

→ first the design requirements are synthesised to create a S/w.

→ second the ~~des~~ synthesised designed is analysed. So that requirements are implemented in design

→ Third & most important part of this iterative process is the decision making. Decision making is based on current & Past knowledge

→ "current" knowledge
  → the requirements
  → the design as created so far

→ "Past" knowledge
  → the technology avaible
  → what has worked well in the Past
  → S/w design Principles & "best Practices".

→ On the basis of decision making Statement, the best solution among the all the Possible solutions are selected.

→ If the best solution is not achieved then again start step ④ i.e. the iterative loop.

→ After the iterative design loop.
At the highest level, the design may be optimized to make best use of the available resources, constraints & expected use.

→ If the design is not optimized than repeat the design loop of iteration.

→ After optimization the evaluation of the design is done. By this process it is checked that user req's are properly implemented in design or not. If not the design constraints & criteria is again checked the repeat the steps from ② to ⑥

→ In the last step the reports are generated with the help of drawings & diagrams that communicate with the users.

Ref. :- R9

7.8 Effective Modular Design :-

A module is an essential part of any modular design.

Definition of Module :-

"A software is divided into separately named & addressable components, sometimes called modules, that are integrated to satisfy problem requirements"

Definition of Modularity :-

"Modularity is the single attribute of software that allows a program to be intellectually manageable".

A system is considered modular if it consists of discreet components so that each component can be implemented separately, and a change to one component has minimal impact on other components.

→ Modularity is a clealy a desirable property in a system.

→ Modularity helps in system debugging,

→ Isolating the system problem to a component is easier if the system is modular.

→ Modular system can be easily built by "Putting its modules together".

→ Modularity reduces the complexty in s/w development

→ Some important criterias that will lead to design the effective modularity.

1) Module. Level concepts

2) Functional Independence

3) Cohesion

4) Coupling

1) Module-Level Concepts :-

→ A module is a logically separable part of a program

→ It is a program unit that is unique & identifiable with respect to compiling & loading.

→ It terms of programming language, a module consist of a function, a procedure, a process or a package.

→ In systems using functional abstraction, a module usually a procedure of function or a collection of these.

→ To produce modular designs, some criteria must be used to select modules so that modules are

Solvable & modifiable separately.

2) Functional Independence :-

→ Functional independence is achieved by developing modules with "single-minded" function & "aversion" (dislike or opposite) to excessive interaction with other modules.

→ Stated in another way, functional independence is to design s/w so that each module addresses a specific sub-req's and has a simple interface, when viewed from other parts of the program structure.

→ Functional independence is important because s/w with effective modularity, i.e. independent modules, is easier to develop because functions may be compartmentalized (divided into categories) & interfaces are simplified.

→ Independent modules are easier to maintain (& test) because secondary effects caused by the design or code modification are limited, error propogation is reduced, & reusable modules are possible.

→ Functional independence is a key to good design, & design is the key to s/w quality

3) Cohession:-

Definition:- "Cohesion is a measure that defines the degree of intra-dependability within element of a module. The greater the cohesion, the better is the program design"

Cohesion of a module represents how tightly the internal elements of the module are bound to one another.

cohesion capture the concept of intramodule bonding.

There are several levels of cohesion:

Worst (low cohesion)

1) Coincidental cohesion
2) Logical cohesion
3) Temporal cohesion
4) Procedural cohesion
5) Communicational cohesion
6) Sequential cohesion
7) Functional cohesion

↓

Best (High cohesion

[illegible]

1) <u>Coincidental Cohesion:-</u>

→ The worst degree of cohesion is coincidental which is placed at the lowest level.

→ Coincidental cohesion occurs when there is no meaningful relationship among the elements of a module.

→ It can occur if an existing program is "modularized" by breaking it in to pieces & making different pieces of modules.

→ If a module is created to save duplicate code by combining some part of code that occurs at many different places, the module is likely to have coincidental cohesion.

→ So the statements in the module have no relationship with each other.

→ ex:- a module that checks a user's security classification & also prints this week's payroll is coincidentally cohesive.

2) <u>logical cohesion</u>:- Elements perform similar activities as selected from outside module. i.e. by a flag that selects operations to perform i.e. body of function is one huge if-else/switch case on operation flag.

2) **Logical Cohesion :-** similar functions such as i/p, error handling etc. put together. Functions fall in same logical class.

→ A module has logical cohesion if there is some logical relationship b/w the elements of a module, & the elements perform functions that fall in the same logical class. may pass a flag to determine which one's executed

ex:-

→ ex:- A module that performs all the i/p's or all the o/p's. In such a situation, if we want to i/p or o/p a particular record, we have to somehow convey this, to the module

→ This will be done by passing some kind of special status flag, which will be used to determine what statements to execute in the module.

→ It results in hybrid information flow b/w different elements of a module, which leads to worst form of logical cohesion.

3) **Temporal Cohesion :-**

→ Temporal cohesion is the same as logical cohesion, except that the elements are also related in time & are executed together.

→ Modules that perform activities like "initialization", "Clean-up", & "termination" are usually temporally bound (time bound).

→ Even though the elements in a temporally bound module are logically related;

→ Temporal cohesion is higher than logical cohesion, because the elements are all executed together (simultaneously).

4) Procedural Cohesion:-

→ A procedurally cohesive module contains elements that belongs to a common procedural unit & just to follow a sequential order

→ ex:- a loop or a sequence of a decision statements in a module may be combined to form a separate module.

5) Communicational Cohesion:-

Communicational cohesion has elements that are related by a reference to the same input or output data.

i.e. in a communicationally bound module, the elements a together because the operate on the same input or output data.

→ ex:- a module to "Print & Punch record".

→ Communicationally cohesive modules may perform more than one function.

→ Communicational cohesion is sufficienty high, as to be generally acceptable.

6) Sequential cohesion:-

→ If the o/p from one part of module is i/p to the next part, the module hase sequential cohesion.

→ All the elements are related in such a way in a module so that the o/p of one forms the i/p to another.

→ But the sequential cohesion doesn't Provide any guidelines on how to combine elements of a module.

→ A sequentially bound module may contain several functions or parts of different functions.

Name of Lecturer: Abhishek Jain

7) Functional Cohesion :-

Functional cohesion is the Strongest cohesion.

Here all the element of the module are related to performing a single function.

Every processing elements is essential to the performance of a single function & all essential components are contained in one module.

ex:- Functions lik "computer square root" & "sort the array".

4) Coupling :-

Definition :- "coupling is a measure that defines the level of inter-dependability among modules of a program".

It tells at what level the modules interfere & interact with each other. The lower the coupling the better ... the program

There are sin level of coupling namely:

Best (Lower coupling)
↑

1) Data coupling
2) Stamp coupling
3) Control coupling
4) Enternal coupling
5) Common coupling
6) Content coupling

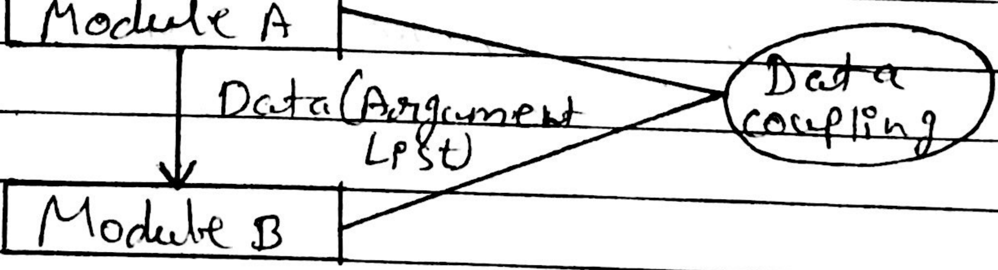
Worst (Higher coupling)

1) Data coupling :-

→ The dependency b/w module A & B is sai to be data coupled if their dependen is based on the fact they communicat by passing data (argument list) only. Other then communicating through data, the too modules are independent.

→ Data coupling is simplear & does not make much error.

→ Data coupling is lowesst coupling b/w modu

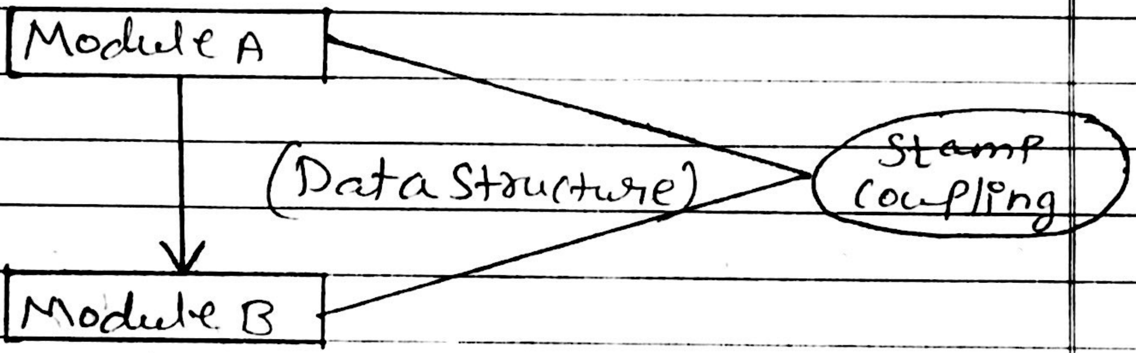


2) Stamp Coupling :-

→ Stamp coupling is a variation of the data coupling.

Global variables are shared selectively

→ Stamp coupling occurs when a data structure is used to pass information between module A to module B.



Stamp Coupling

3) Control Coupling :-

→ Module A & B are said to be control coupled if they communicate by passing of control function.

→ In this one module passes parameters to control the activity of another module

→ This is usually accomplished by means of flags that are set by one module & reacted upon by the dependent module.

4) **External Coupling :-**

→ External coupling is the coupling in which high coupling occurs.

→ Modules are tied to an environment external to s/w.

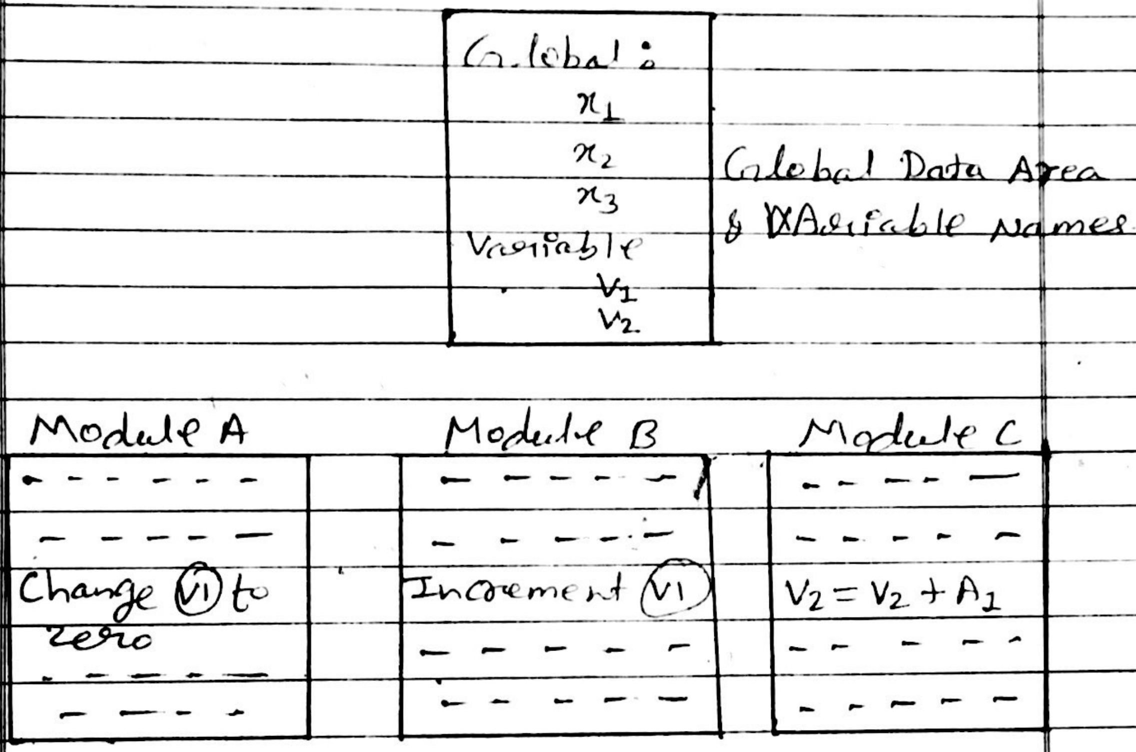
→ It is essential but should be limited to small number of modules with in a structure.

5) **Common Coupling :-**

In common coupling module A & Module B can access a shared data (global data area).

Global data areas are commonly found in programming languages.

Making a change to the common data means tracking back to all modules that access that data to evaluate the effect of that change



Common Coupling

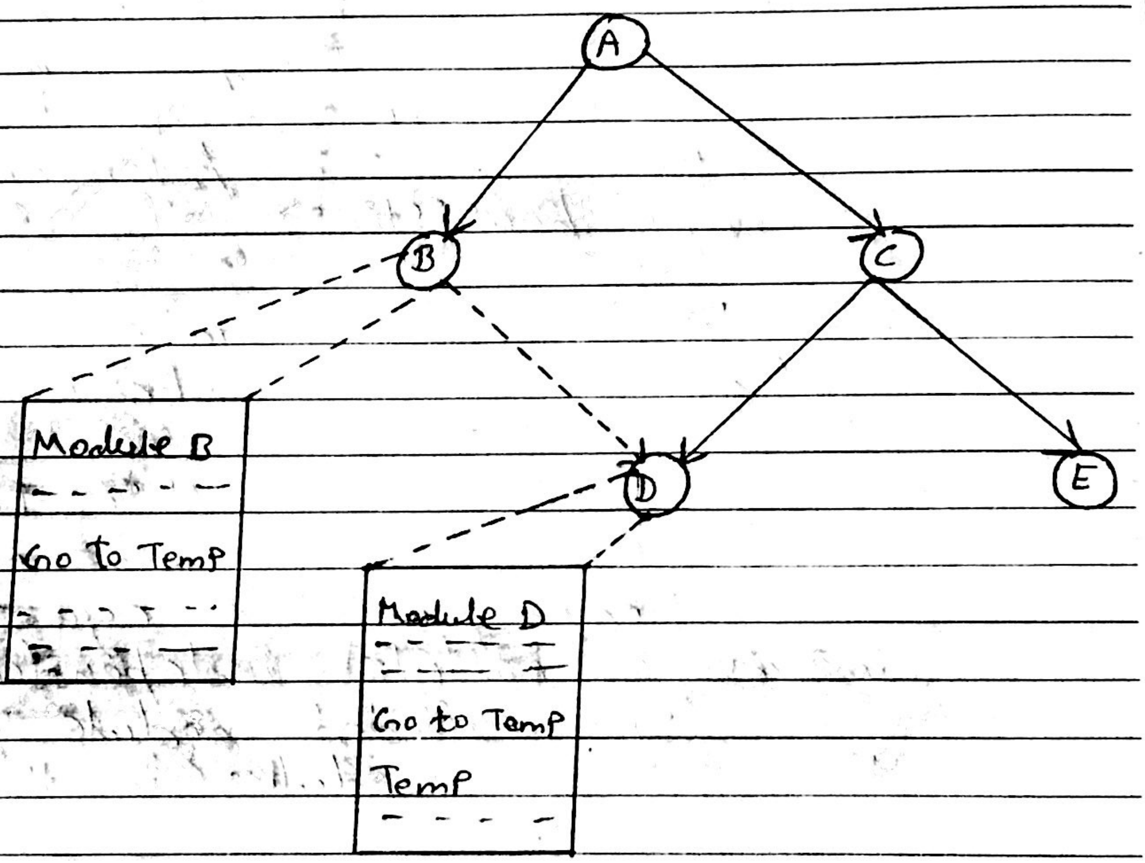
6) Content Coupling :-

→ Content Coupling is the least desirable coupling.

→ Content coupling occurs when module A changes data of module B or when Control is passed from one modcule to the the middle of another.

→ When one module actually modifies another module, then the modified module is completely dependend on the modifying one

ex:-



Content Coupling.

→ Module B branches into D, even though D is supposed to be under the control of C.

Ref. :- R10,R6

4.9 Effective Modular Design Heuristics:-

Once the program structure has been developed, effective modularity can be achieved by applying the design concepts. The program structure can be manipulated according to the following set of heuristics:

1) Evaluate the First Iteration (reduce coupling & increase cohesion):-

→ One the program structure has been developed, modules may be exploded our imploded with an eye toward improving module independence.

→ An exploded module becomes (result in) two or more modules in the final program structure.

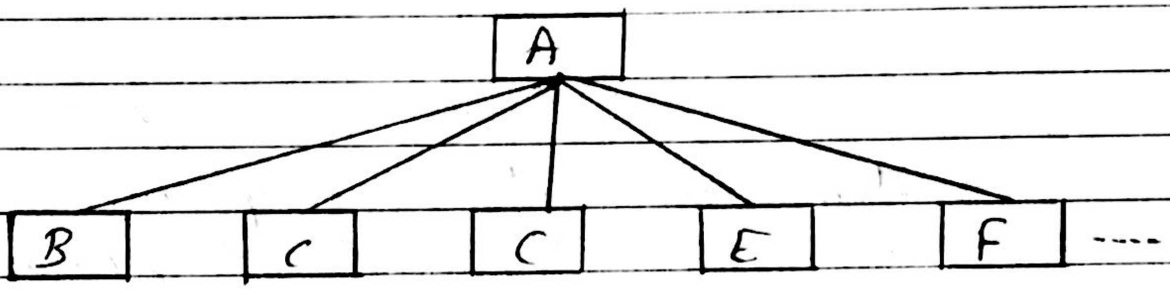
→ An imploded module is the result of combining the processing implied by two or more modules.

→ An exploded module often results when common processing exists in two ore more modules & can be redifined as a separate cohessive module.

→ When high coupling is expected. modules can be sometimes be emploded to reduce passage of control, reference to global data & interface complexity.

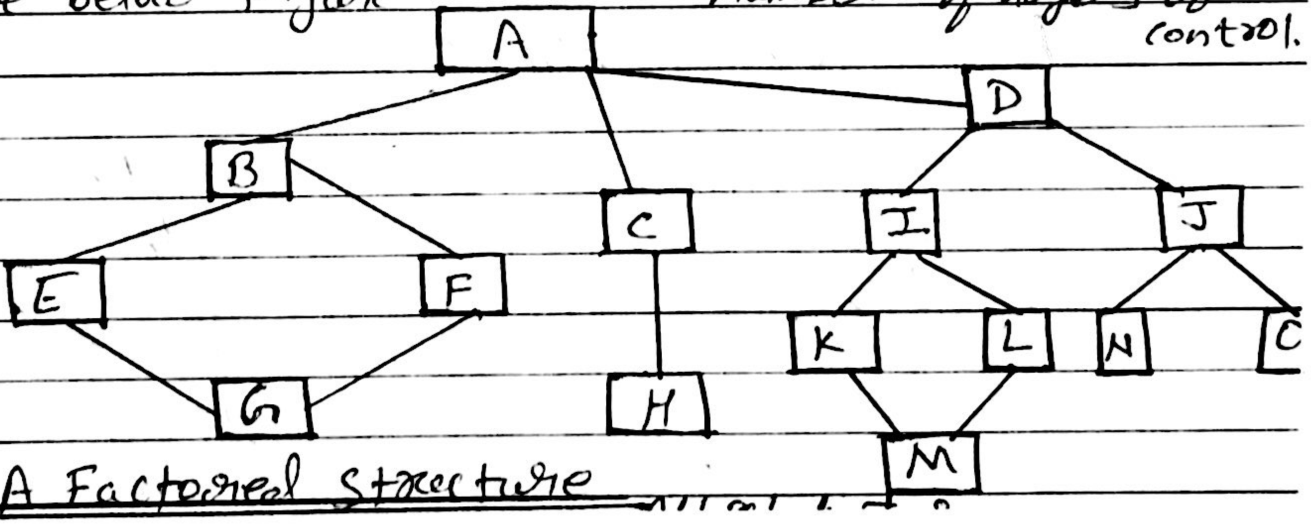
2) Minimize the Structure with High Fan-Out:

→ Try to avoid the Pancaked Structure in which all the modules are under the control of a single module. A Pancaked structure is drawn below.



A Pancaked Structure.

→ To avoid such type of Pancaked Structure a more reasonable distribution of control is required. The below Figure shows a number of layers of control.



A Factored Structure

3) Keep the scope of EFFect of a module within the scope of control of that module :-

→ The scope of a particular module(I) as all modules that are affected by a decision made in this module(I). The scope of control of I is all the modules that are subordinate of module(I).

in last fig

4) Reduce Complexity :-

→ Evaluate module interfaces to reduce complexity & redundancy & improve consistency.

→ Interface designing should be like so that information can be easily passed & consistent with the function of a module.

5) Predictable Function :-

→ Define module whose function is predictable but avoid modules that are overly restrictive.

A module is predictable when it can be treated as a black box; i.e., the same external data will be produced regardless of internal processing details. Much restricted modu Should be avoided.

6) Strive for "controlled entry" modules by avoiding "Pathological connections" :-

This design heuristic warns against con Coupling.

Software is easy to understand & there easier to maintain When module Interf are constrained & collected.

Pathological connection refers to bran o.r. references into the middle of module.

Ref :- R11,R12,

4.10 Design Methods :-

According to Freeman

"Design is an activity concern with making major decisions, often of a struct nature. It Shares with programming a co for abstracting information representation processing sequences, but the level of d is quite different at the extremes. Design builds coherent, well-Planned representation of Programs that concentrate on the interrelationships of parts at the higher level & the logical operations involved at the lower level".

Following design methods are as follows.

1) Data Design

2) Architectural design

3) Interface Design

4) Component-Level Design

5) Deployment-Level Design.

4.11 1) Data Design:-

→ Data design is also called as the Data architecturing.

→ Data design is the first of three-design activities that are conducted during S/w engg.

→ Data design creates a model of data

& information that represents a high level of abstraction.

→ Data design created by transforming the analysis information model (data dictionary & ERD) into data structures required to implement the Software.

→ Part of the Data Design may occur in conjunction with the design of S/w architecture.

→ More detailed data design occurs as ea software component is designed.

1.12 2) Architectural Design :-

→ The large systems are always decomposed into subsystems that provide some related set of services.

→ The main objective of architectural design is to develop a modular program structur & represent the control relationship b/ modules.

→ It also defines the relationships among th major structural elements of the software,

It defines

→ the "design Patterns" that can be used to achieved the req's that have been designed for the system.

→ It defines the constraints that affect the way in which the architectural patterns can be applied.

→ Architectural design is derived from the system specification, the analysis model, & the subsystem interactions defined in the analysis model (DFD).

4.13 3) Interface Design:-

→ It describes how the s/w elements communicate with each other, with other systems, & with human users.

→ The Data Flow Diagram (DFD) & Control Flow Diagram (CFD) Provide much of the necessary information required.

4.14 4) Component-Level Design :-

→ It is created by transforming the structura elements defined by the s/w architecture into Procedural descriptions of s/w component using information obtained from the Process specification (PSPEC), Control Specification (CSPEC), & State Transition Diagra (STD).

4.15 5) Deployment-Level Design :-

It indicates how s/w functionality & subsystems will be allocated within the physical computing environment that will support the s/w.

Ref :- R1; P.No.:-274-279, R14

4.16 Design Documentation :-

Definition :- "A software Design Documentation (SDD) is a written description of a s/w product, that a s/w designer writes in order to give s/w development team overall guidence of the architecture of the s/w project".

→ A design document is required to co-ordinate a large team under a single vision.

→ A design document needs to be a stable reference, outlining all Parts of the s/w & how they will work.

→ The document is commanded to give a fairly complete description, While maintaining a high level view of the s/w.

Document Outline;

1) Introduction:-

1.1 Purpose

1.2 Scope

1.3 Definitions & Acronyms

2) Reference Documents:-

2.1 Existing s/w Documentation

2.2 System Documentation

2.3 Vendor (H/w or s/w) documents

2.4 Technical Reference.

3) Design Considerations :-

3.1 Assumptions & Dependencies.

3.1.1 Related s/w or H/w

3.1.2 Operating Systems

3.1.3 End-users Characteristics

3.1.4 Possible and/or probable changes in functionality.

4) Architectural Strategies :-

4.1 Reuse of existing s/w component to design a Syste

4.2 Future Plans for extending or enhancing the s/w

4.3 User interface Paradigms (Sys. i/p or o/p Models,

4.4 H/w &/or s/w interface Paradigms.

4.5 Error detection & recovery.

4.6 Memory management policies.

4.7 Distributed data or control over a N/w.

5) Modules :- for each module.

5.1 Processing Narrative

5.2 Interface Description

5.3 Design Language (or other) descriptions

5.4 Modules used

5.5 Data organizations.

5.6 Comments.

6) File Structure & Global data :-

6.1 External File Structure

6.1.1 Logical Structure

6.1.2 Logical Record Description

6.1.3 Access Method.

6.2 Global Data

6.3 File & Data Cross Reference.

7) Test Provisions :-

7.1 Test Guidelines

7.2 Integration strategy

7.3 Special Considerations (Test cases)

8) Requirements Cross Reference :-

9) Glossary :-

10) Bibliography :-

Ref :- R6, R15, R16

4.17 Programming Languages :-

Definition :- "A Programming language is a formal Constructed language designed to communicate instructions to a machine".

Definition :- "A Programming language is a notation for writing

Programs, which are specifications of a computation or algorithm".

→ A Programming languages can be used to create Programs to control the behavior of a machine to express algorithms

→ With the help of computer programming languag a computer performing some kind of computat-on & Possibly control electronic devices such as Printer, robots, disk drives etc.

→ Programming languages differ from natural languages, in that natural language are only used for interaction b/w people, while programming languages also allow humans to communicate instructions to machine.

Ref. :- R17

Types of Programming languages.

1) Procedural languages
2) Non-Procedural languages
3) Object oriented languages.
4) Visual languages
5) Functional Programming languages.
6) Logic Programming Languages
7) Generic languages-

Name of Lecturer : Abhishek Jain.

8) Declarative languages
9) Imperative languages.
10) Fourth Generation languages.

1) Procedural Languages:-

→ Procedural Prog. languages are the conventional Prog. languages in whic programs are decomposed into finite Steps that Perform complex operations.

→ A Procedural program is composed of one or more units or modules, either user coded or provided by a code library.

→ Each module is composed of one or more Procedures, also called a function, routine, subroutine or method depending the language.

ex:- ADA, BASIC, C, C++, COBOL, Pascal etc

2) Non-Procedural Languages:-

→ It include the "what" Part, but execlude the "how" part.

→ The programmer follows What to be accomplishe but does not follow how to accomplish the task.

→ Programs in such languages do not state exactly how a result is to be computed but rather describe the form of a result

ex:- SQL, dBase, Paradox, ADS etc.

3) Object-Oriented Languages :-

OO Languages are based on definition of classes & objects. Instaces of classes are created by the program as needed during program execution.

ex:- Small Talk, C++, Java etc.

4) Visual Languages :-

→ In this languages users can specify programs two or more dimensinal array, instead of as one-dimensional text strings, via graphical layout of various types.

→ These languages are very Popular now a days, because they are very easy to learn & understand.

→ ex:- CODE, Lava, Prograph, Toon Talk etc.

5) Functional Programming Languages:-

→ Functional Programming languages define Programs & subroutines as mathematical functions.

→ Many so-called functional languages are "impure", containing imperitive features.

→ ex:- Pure → Miranda, Mercury, curry etc.
impure → C#, J, LISP, Python etc.

6) Logic Programming Languages:-

→ "Logic Programming allows a Programmer to describe the logical structure of a Problem rather than describe how a Computer is to go about solving it".

→ It uses a form of symbolic logic as a programming language.

→ ex:- Prolog, FUNLOG, LOGLISP, SASL etc.

7) Generic Languages :-

→ Generic Languages based on the definition of generic modules that may be initiated either at compile time or run time.

→ The generic programming paradigm does not exist in Isolation.

→ It can exist jointly with OO Programming or with Functional programming.

→ ex:- Eiffel, ML, Ada, etc.

8) Declarative Languages :-

→ Declarative languages describe Problem rather than defining a solution.

→ Such Programs are closer to a specification than a traditional implementation.

→ Many Logic & Rule based languages are the Declarative languages.

→ ex:- Prolog, SQL, Oz, MetaPost etc.

9) Imperative Languages:-

→ Imperative languages are characterized by three concepts: Variables, Assignment & sequencing.

→ Imperative Languages are those in which expressions are computed & accordingly their results are assigned (Stored) in the variables.

→ Imperative languages are also called as State-Based language & Assignment oriented Language.

→ ex:- ALGOL, Modula-2, C, C++ etc.

10) Fourth Generation Languages:-

→ Fourth Generation Languages are High-Level languages built around database Systems. They are generally used in Commercial environment.

→ ex:- SQL, LINC4GL, Progress 4GL, ABAP etc.

.18 / <u>Programming Language concepts :-</u>

To understand Programming languages more efficiently; various Programming concepts are as follows.

1) Programming Language qualities.

2) Features of Programming Languages.

3) Selection of Programming Languages.

1.19 1) <u>Programmig Language Qualities :-</u>

Programming qualities help to develop an effective & efficient code. Some of the qualities are,

Three important aspect of s/w which leads to quality are as follows.

1) s/w Reliability

2) s/w Maintainability

3) s/w Efficiency.

Major Programming Language Qualities and their Description.

Goal	Quality	Major Issues.
1. Reliability	1- Writability	• Write a Program in general, easy & natural f • Mention the details completely to solve the Problem. • A Subjective criterion • Much followed in higher-level languag • Leads to less errors if concentration is more on Problems.
	2. Readability	• Follow the logic of the Program. • Find the errors in the Program. • A subjective criterion.
	3. Simplicity	• Easy to use and understand • Makes programmer Confident • Easily expressing the algorithms
	4. Safety	• Not to use features to make harmful Programs • Harmful Features leads errors.

Goal	Quality	Major Issues
	5. Robustness	• Handles the unexpected or undesired ev • Uses Exception Handling • Exhibit normal behavior in undesi Conditions.
Maintainability	1. Readability	• Similar issues as of readability in reliability. • Exhibit good quality readability while modifying the Program
	2. Simplicity	• Similar issues as of simplicity in reliability. • Easy to understand, use & grasp.
	3. Factoring	• Combine similar & related features into one • Increases readability & modifiability
	4. Locality	• make small, Local parts of the program • Increases modifiability • makes changes easily in complex systems.
Efficiency.		• Requires execution speed, space & effort required initially & effort required in maintenance to measure the efficiency. • Tries to save space & improve speed. • Allows certain optimizations to be applied by compiler.

4.20 2) Features of Programming Languages:-

Various Features of programming languages are.

1. Variable name & scope
2. Pre processar
3. Named Constants
4. Binding, scope & Extent
5. Data types
6. Available Operations
7. Data Structures
8. Recursion
9. Data Abstraction
10. Procedural Abstraction
11. Control Abstraction.
12. Exception Handling
13. Support for OO Approaches
14. Language Library
15. Compilers
16. Debuggers
17. Memory Allocations
18. File Handling Utilities
19. System Utilities
20. Concurrency mechanisms
21. Parallel considerations
22. Real-Time considerations

4.21 3) Selection of Programming Languages:-

The art of choosing a language is to start with the Problem, decide its requirements are, & their relative importance, since it will probably be impossible to satisfy them all equally well (with a single language), available languages should be measured against a list of req's.

To Select a Programming language then a following Characteristics are desirable.

1) Ease ~~to~~ of design to Code
2) Compiler Efficiency
3) Source code portability
4) Availability of Development Tools
5) Maintainability
6) Uniformity.
7) Knowledge of S/w Development Staff

Ref :- R17

22 Programming Guidelines :-

Several guidelines that apply to Programmin in general, regardless of the language. Programming guidelines are

1) Localizing Input & output

2) Including Pseudocode

3) Revising & Rewriting, not Patching

4) Reuse.

1) Localizing Input & output :-

→ Those Parts of a program that read input or generate output are highly specialized & reflect characteristics of the underlying H/w & S/w.

→ Because of this dependence, the program sections performing i/p & o/p functions are sometimes difficult to test.

→ There are some sections is likely to change if the H/w or S/w is modified.

→ It is desirable to localize these sections in components separate from the rest of the code.

Ref :- R19

2) Including Pseudocode :-

Definition of Pseudocode :-

"Pseudocode is a detailed & readable description of what a computer program or algorithm must do, expressed in a formally-styled natural language rather than ~~in~~ a programming language".

→ Psedocode is sometimes used as a detailed step in the process of developing a Program.

→ Pseudocode can be used to adapt the desig to your chosen language.

→ The Pseudocode has acted as a framework on which the code is to be constructed.

→ catching errors at the Pseudocode Stage is less costly than catching them later in the development process.

→ once the Pseudocode is accepted, it is rewritt using the vocabulary & syntax of a programming language.

Ref.:- R20

3) Revising & Rewriting, Not Patching :-

→ When writing code for the s/w development, we often write a rough draft.

→ Then it is carefully revised & rewritten until satisfied results are not achieved.

→ The design is reexamined to see whether the problems are related to the design part or in the translation of design to code.

Name of Lecturer : Abhishek Jain

4) **Reuse:-** As such there are two kinds of reuse:

i) Producer Reuse

ii) Consumer Reuse.

i) **Producer Reuse:-** It is that where we are creating components designed to be reused in subsequent applications.

ii) **Consumer Reuse:-** In this we are using components that were originally developed for other projects.

Some of the characteristics to reuse as a consumer.

→ Does the component perform the function or provide the data you need?

→ Is less modification required than building the component from scratch?

→ Is the component well documented, so that you can understand it without having to verify its implementation line-by-line?

→ Is there a complete record of the

Component's test & revision history, so that you can be certain that it contains no faults?

Ref. :- R19

1 Structured Programming Concepts :-

Definition :- "Structured programming is a Programming paradigm aimed at improving the clearity, quality & development time of a computer Program by making extensive use of subroutines, block structures & for and while loops & avoid the use of goto statements which is difficult to follow & maintain".

24 Elements of Structured Programming :-

1) Control structures.

2) Subroutines.

3) Blocks.

1) Control Structures :-

Following the structured programming, all program are seen as composed of three control structures

→ "Sequence", Orderead Statements or Subroutines executed in sequence.

→ "Selection", one or a no. of Statements is executed depending on the state of the Program. This is usually expressed with keyword such as if... then,...else... endif.

→ "Iteration", a statement or block is executed until the program reaches a certain state. This is usually expressed with keywords such as While, repeat, for or do... until

2) **Subroutines:-** Callable units such as Procedures, functions, methods or subPrograms are used to allow a sequence to be referred to by a Single statement.

3) **Blocks:-** Blocks are used to enable groups of statements to be treated as if they were one statement.

Ref:- R21

4.25 Programming Style :-

Definition :- "Programming style is a set of rules or guidelines used when writing the source code for a computer program. It is often claimed that following a particular programming style will help programmers to read & understand source code conforming to the style, & help to avoid introducing errors".

→ Programming styles are often designed for a specific language.

Ref. :- R22

4.26 Programming Style Rules :-

1) Use of consistent & meaningful variables nam

2) Use standard control structures.

3) Use goto in a disciplined manner (don't use if not necessary).

4) Introduce user defined data types.

5) Isolate Machine dependencies in a few routin

6) Hide data Stouctures behind access function.

7) Make effective use of Comment Statements.

8) Provide Standard documentation Prologues (Introduction to Preface) for each SubPrograms and/or compilation unit.

9) Use Function SubProgram & Procedure SubProgram Appropriately.

10) Carefully Examine Routines.

11) Use identation (Systematic way writing a Program), Parenthesis, Blank, Spaces, Blank Lines & Borders Around Comment Blocks to Enhance Readability.

Ref.:- R23

4.27 Programming Style Metrics:-

Definition:- "In S/w development, a metric is the measurement of a Particular

Characteristic of a Program's performance efficiency".

Ref. :- R24

Some of the style metrics are.

1) **Module length :-** Average length of modules. It is measured for non-comments & non-blank lines.

2) **Identifier Length :-** An identifier is measured in characters. It is the average length of the user-defined identifiers

3) **Comment Lines :-** Comment lines can be measured as the Percentage of the Comment lines in total lines of Code of the progra

4) **Identation :-**

$$\text{Identation Measurement} = \frac{\text{Initial SPaces}}{\text{Total No. of characters.}}$$

5) **Blank Lines :-** It is a Percentage of Blank line with the total no. of lines.

6) **Line Length :-** A line length is find out as the average no. of nonblank Characters. in a lin

7) **Constants Definitions :-**

Percentage of all user identifiers that are define as constants.

8) **Reserved Words :-**

In this case, no. of reserved words & standard library functions used are counted.

9) **Include Files :-**

The no. of files that are included in a Program is measured.

10) **Goto's :-** The no. of occurences of the goto statement provides the total no. of goto statements as its measurement.

[Ref :- R24]

P.T.O. →

.28 **What is the code :-**

Definitions :- "It is a practical implementation of data or instructions in a computer programming language" or the set of such instructions".

Ref. :- R25

OR

"In programming, code is a term used for both the statements written in a particular programming language.- the source code, and a term for the source code after it has been processed by a compiler & made ready to run in the computer - the object code".

Ref. :- R26

.29 **Code Guidelines :-**

Rules & guidelines to develop an efficient & structured code. With the help of these guidelines, detailed design is translated into code.

Before you write one line of code, be sure you:

1) Understanding the problem youare trying to solve.

2) Understand basic design principles & concepts.

3) Pick a Programming language that meets the needs of the Sw to be built & the environment in which it will operate.

4) Create a set of unit tests that will be applied once the component You code is completed.

As You begin Writing Code, be sure you:

1) Convert your algorithm to code by following structured programming practice.

2) Select the data structures, that will meet the needs of the design.

3) Keep conditional logic as simple as Possible.

4) Create nested loops in a way that makes them easily testable.

5) Select meaningful variable names & follow other local coding standards.

6) Write code i.e. self documenting.

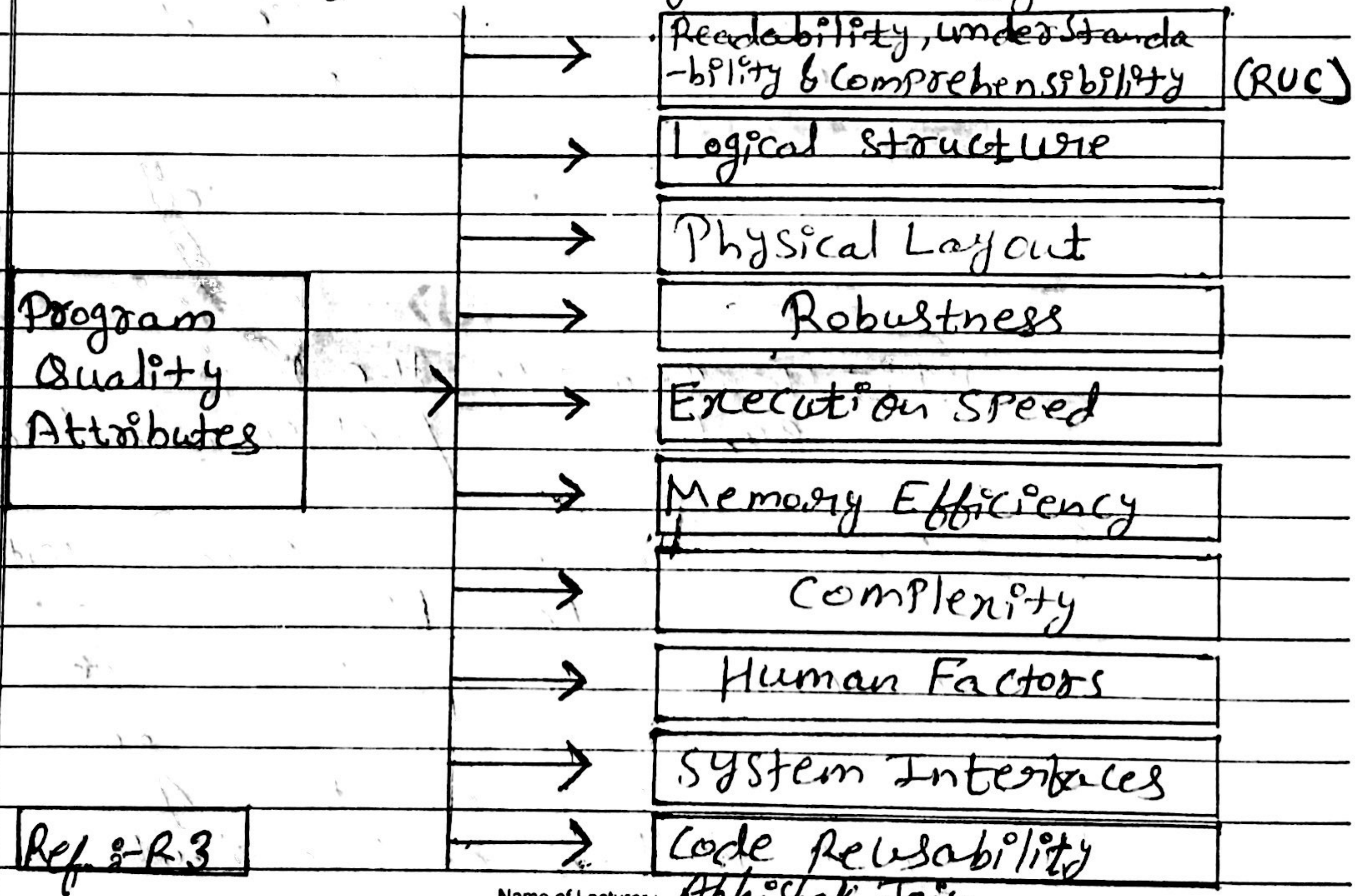
7) Create a visual layout (ex: identation & blank lines) the aids understanding.

8) Don't mix data types, even if the language allows it.

Ref. :- R1; P.No:-145

1.30 Program Quality :-

Definitions:- "Program quality is the collection of attributes of a program so that it makes itself able enough to satisfy the needs".



Ref :- R3

Name of Lecturer : Abhishek Jain

Scanned by CamScanner

[4.31] Program Quality Quantification:-

Program Quality Quantification is is the measurement of quality of Program.

Program Quality is measured by measuring all program quality attributes one-by-one.

To measure the program quality attributes there is two-step procedure required.

Step 1:- Assign the raw Score Points & weight Factors to each of the ten attributes.

Step 2:- Calculate the Composite quality score.

Step 1:- To quantify the Program quality, two factors are discussed:-

a) **Raw Score:-** Important issue concerned with Raw ~~S~~Score(RS) factor are

1) A raw score is the Points given to each of the ten attributes to show that the Program is consisting of "how much" Points of each attribute. ex:- Consider a program in which robustness has 60 Points. Then we can say that this Program is almost 60% robust

2) The raw Score has the points range from 1 to 100.

3) A team of experienced members are required to assign the raw Score to each attribute.

4) To assign raw score points, team checks the Complete Program, its performance, its effectivenes its Capability, & used guidelines, standards, rules etc.

5) No default value is used to assign the raw score points.

6) The raw Scores of 1 to 100 are categorized in four section.

Category	Point Ranges of Raw Score
Best	90 or Above
Good	Between 80 to 90
Average	Between 70 to 100
Low	Below 70

b) Weight (W) :- Some related issue in weighting (WF) factor are

1) A WF refers to the importance ~~or~~ Priority of a particular attribute.

2) It has a range of 0 to 10.

3) Each of the attribute must consist of one WF.

4) WF of each attribute depends on the importance of the attribute in the S/w product.

5) Default weight value for each of the ten attributes is 1.

6) Weigh value assignment to each of the attribute depends upon the particular application area of S/w. So, these weight value assignments may vary from application to application.

7) Weight factors are fixed for a given module, subprogram or a complete S/w product

Using these two major factors values, various program quality attributes are measured.

Step 2 :- Calculate the Composite Quality score

Composite Quality Score is the computation of program quality which is calculated by the total weighted score by the Total weight. Its eqn. is.

$$\text{Composite Quality score} = \frac{\text{Total Weighted score}(WS_T}{\text{Total weight }(W_T)}$$

Where,

a) Total weighted score (WS_T) is the sum of the ten weighted scores. so

$$\text{Total Weighted Score}(WS_T) = \sum_{i=1}^{10} WS_i$$

$$= WS_1 + WS_2 + \cdots\cdots + WS_{10}$$

A weighted score is the product of a raw score & a weight factor for each of the ten attributes. So,

$$\text{Weight Score }(WS) = \text{Raw Score} \times \text{Weight}$$

for $i = 1$ to 10, this eq^n is written as

$$WS_i = RS_i \times W_i \ [\text{for } i = 1 \text{ to } 10]$$

b) Total weight is the sum of the ten weights are,

$$\text{Total weight }(W_T) = \sum_{i=1}^{10} W_i$$

$$= W_1 + W_2 + \cdots\cdots + W_{10}$$

4.32 Complete Programming Example :-

Programming example describes the following aspects.

1) Top-Level Design Specification :-

We have to follow some design guidelines for solving a problem.

2) Analysis of Preliminary Design :-

We should examine whether any error or fault or problem is existing in top-level design specification & whether this specification follows requirement specification or not.

If any error occurs that should be removed & top-level design specifications must be modified & corrected

3) Main Data Structures :-

Main data structures are focuses on inter values inserted & no complex data structure is required. So we will concentrate on these integer data types only.

4) High-Level Program Structures :-

After developing top-level design specifications, analyzing them & mentioning requisired data structures, now we will see high-level program structure, which may contain some functions & procedure calls definitions in itself.

5) Detailed Design Description :-

We have to describe the detailed design of this program structure & mention the complete function structure.

6) Program Development Process :-

Now the detailed design description must be converted into the code.

This code will then be tested during testing activity for checking faults, debugging & correct errors.

Ref :- R 3, R24