SOFTWARE ENGINEERING

Page No.:
Date: / /

Unit - 3

ONE SHOT + 3 PYQs

⇒ Topics :

☆ Basic concept of software Design
- Architectural Design
- low level Design, Modularization, Design
- structure charts, Pseudo codes, flow chart
- Coupling and Cohesion Measures

☆ Design Strategies
- Function oriented Design
- Object oriented Design
- Top-Down and Bottom-Up Design.

☆ Software Measurement and Metrics:
- Various size oriented Measures: Halstead's software Science
- Function Point (FP) Based Measures
- Cyclomatic Complexity Measures: control flow Graphs

Page No.
Date / /

⇒ Software Design :

- Software design is the process of defining the architecture, components, modules, interfaces, and data for a software system to satisfy specified requirements.
- It is intended to be a blueprint for the construction of the software system.
- Software designing is most creative process, where we actually decide how a problem will be solved.

SRS ⟶ [Design] ⟶ SDD.

⇒ Objective of Software Design :

- Correctness : Software design should be correct as per requirement
- Completeness : The design should have all component like data structures, modules, and external interface, etc.
- Efficiency : Resources should be used efficiently by the program.
- Flexibility : Able to modify on changing needs
- Consistency : There should not be any inconsistency in the design.
- Maintainability : The design should be so simple so that it can be easily maintainable by other designers

Page No.:
Date: / /

⇒ Good Software Design [AKTU - 2021-22]

- The definition of a "good software design" can vary depending on the application being designed.
- For good quality software to be produced, the software design must also be of good quality.

⇒ Characteristic of good software design :-

1. Correctness – A good design should correctly implement all the functionalities identified in the SRS document.
2. Understandability - A good design should be easily understable for which it should be modulars all the modules are arranged in layers
3. Efficiency – A good s/w design should address the resources, time & cost optimization issues
4. Maintainability – A good s/w design should be easy to change whenever a change request is made from the customer side

Page No.
Date / /

⇒ Phases of Software design :-

① The software design process can be decomposed mainly into the following three level of design

1. Interface Design
2. Architectural Design
3. Detailed Design

② Interface Design :-

- Interface design is the specification of the interaction between a system and its environment
- This phase proceeds at a high level of abstraction with respect to the inner workings of the system.
- Attention is focused on the dialogue between the target system and the user, devices and other system with which it interacts.
- The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called agents
- Often the major task of interface design is user interface design especially for systems with complex graphical user interfaces

③ Architectural design :

- Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces and the relationships and interaction between them

- In architectural design the overall structure of the system is chosen, but the internal detail of major component are ignored.

⑥ Detailed design:

- Detailed design is the specification of the internal elments of all major system components, their structure, properties, relationships, processing and often their algorithms and data structure.

- Detailed design consist of a wide range of "detail" because there is usually quite a lot to fill in between the architecture and the code.

⇒ Software Design Principles :-

- There are some principles for good system design.

- Problem partioning / Decomposition : It refer to break down a complex system into component or smaller sub-system.

- Abstraction : It means to describe the external behaviour of that component without bothering with the internal details that produce the behaviour.

- Modularity :- It mean the division of software into sipearate modules which are differently

addressed and are integrated later on it to obtain the complete functional software.

- Top-down & bottom-up approach: The top-down approach start with the identification of the main component and then decomposing them into more detailed sub-components.

- The bottom-up approach begins with the lower details and moves towards upward to form one main component.

⇒ Modularization :-

- Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently.

- Modules are interconnected through their interfaces.
- The interconnection should be simple to avoid costly maintenance.
- Two modules are directly connected if one module can call the other module.

Modularity has several key benefits.

- Testing & Debugging: Since each component is self-contained, you mitigate dependency issues.

- It becomes easy to test each component in isolation by using a mocking or isolation framework.
- Reusability :- If you discover you need the same functionality in a new project, you can package the existing functionality into something reusable by multiple project without copying and pasting the code.
- Extensibility :- Your software now runs as a set of independent component connected by an abstraction layer.

Adv :—

- Smaller components are easier to maintain
- program can be divided based on functional aspects
- Easier because programmer only focus on one small simple problem rather than a large complex problem
- Testing and Debugging easier.
- Easy to isolate bugs and easy to fix bugs

Page No.
Date / /

⇒ Design Structure charts :- [AKTU - 2022-23]

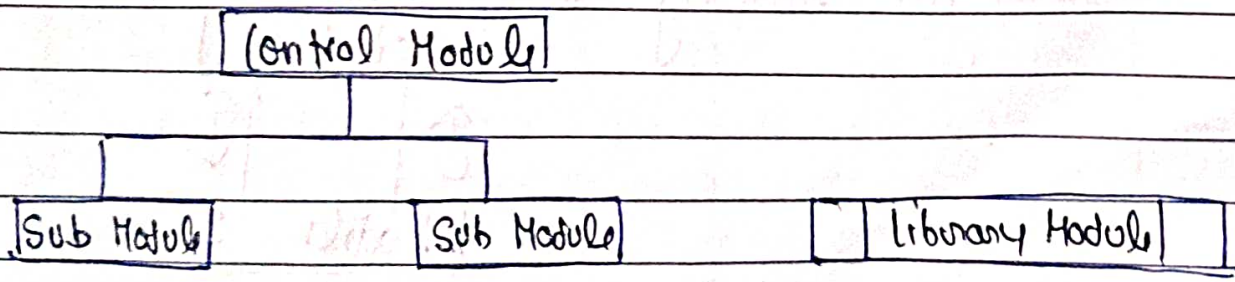
- Structure chart represent hierarchical structure of modules.
- It breaks down the entire system into lowest functional modules, describe functions and sub-functions of each module of a system to a greater detail.
- Structure chart partitions the system into black boxes (functionality of the system is known to the user but inner details are unknown).
- Inputs are given to the black boxes and appropriate outputs are generated.

⇒ Symbol :

- Module : It represent the process or task of the system.

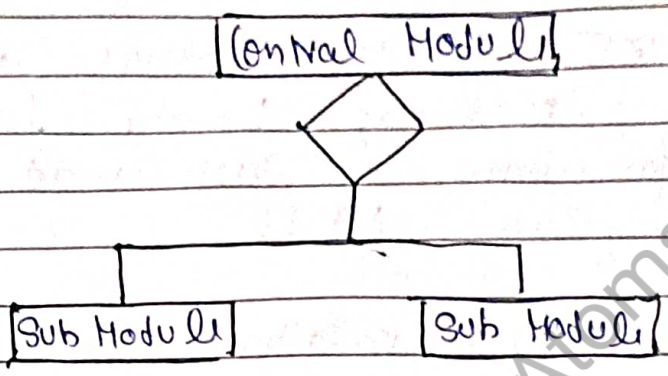
→ It is of three types.

- Control Module : A control module branches to more than one sub module.
- Sub module : Sub Module is a module which is the part (child) of another module.
- Library module : Library module are reusable and invokable from any module.



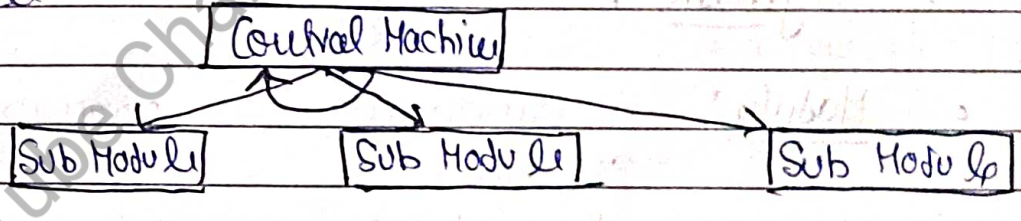
Page No.:
Date : / /

- Conditional Call : It represents that control module can select any of the sub module on the basis of some condition



- Loop (Repetitive call of Module) : It represent the repetitive execution of module by the sub module.

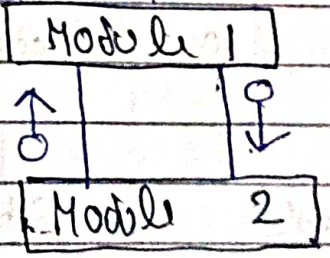
→ A curved arrow represents a loop in the module.



→ All the submodules cover by the loop repeat execution of module.

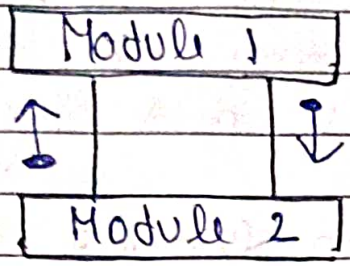
- Data Flow :- It represent the flow of data between the modules.

→ It is represented by a directed arrow with an empty circle at the end.

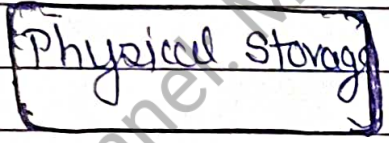


- Control Flow : It represents the flow of control between the modules.

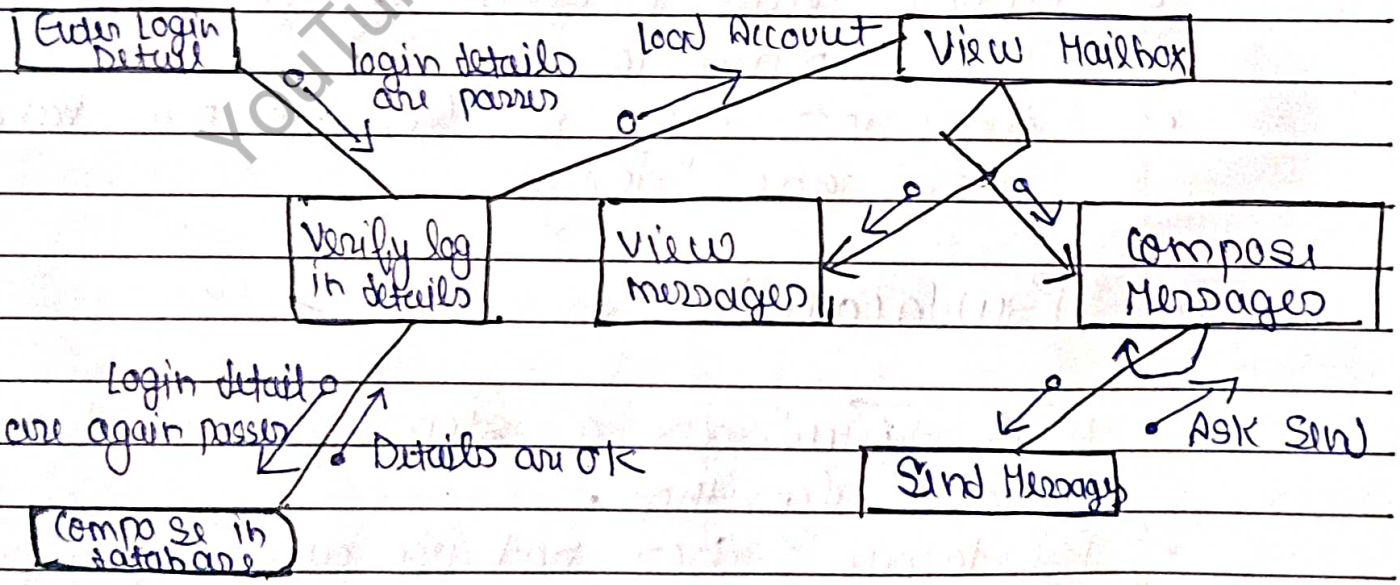
→ It is represented by a directed arrow with a filled circle at the end.



- Physical Storage : It is that where all the information are to be stored.



Example :-
Structure chart for an Email Server



Page No.:
Date: / /

⇒ Types of structure chart

1] Transform centered structure: There type of structure chart are designed for the systems that receives an input which is transformed by a sequence of operation being carried out by one module.

2] Transaction centered structure: There structure describes as a system that processes a number of different types of transaction.

Benefits

- Improved understanding of system architecture
- Simplified communication between team members.
- Easier identification of potential issues and dependencies
- Support for modular and maintainable software design

Pseudocode :-

- It is defined as a step-by-step description of an algorithm.
- Pseudocode does not use any programming language in its representation instead it uses the simple English language text as it is intended for human

understanding rather than machine reading.

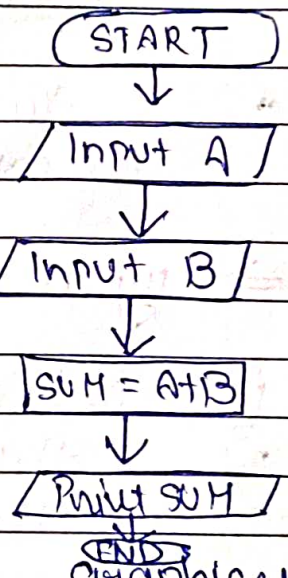
- Pseudocode is the intermediate state between an idea and its implementation in a high level language.
- Pseudocode helps developer plan and discuss algorithm, making it easier to understand and translate into actual code later in the software development process

Ex :- Find the sum of two numbers

Pseudocode :

```
Begin
   Input A
   Input B
   Compute SUM = A+B
   Print SUM
End.
```

Flowchart



Flowchart :—

- Flowchart are nothing but the graphical representation of the data or the algorithm for a better understanding of the code visually
- It displays step-by-step solutions to a problem, algorithm, or process.

Page No.:
Date : / /

⇒ Low level Design :-

- LLD is a phase in the software development process where detailed system components and their interactions are specified.
- Low-level design refers to the process of specifying and defining the detailed design of software system
- It provides the foundation for high-level design, which defines a system overall architecture design
- In this phase, the actual software component are designed.

⇒ Types of low level design :-

1. Modularization
2. Structure chart
3. Pseudocode.

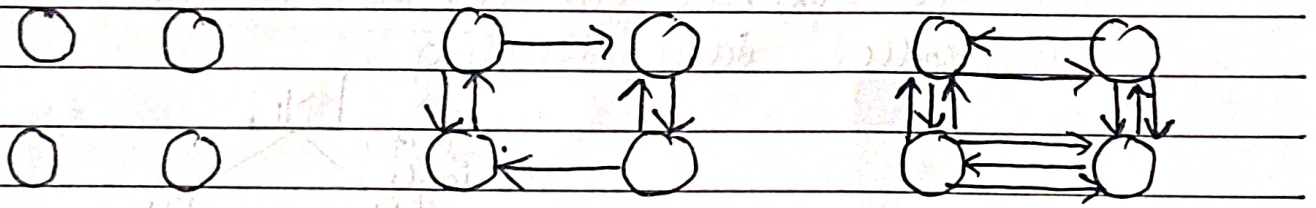
Page No.
Date: / /

⇒ Coupling and Cohesion Measures [AKTU-21/22]

⇒ Coupling and cohesion are two parameters on which we can understand the quality of modularity in the design of a software.

⇒ Coupling

- The coupling is the degree of interdependence between software modules.
- Two modules that are tightly coupled are strongly dependent on each other.
- However, two modules that are loosely coupled are not dependent on each other.
- Uncoupled modules have no interdependence at all within them.



Uncoupled: no dependencies (a)

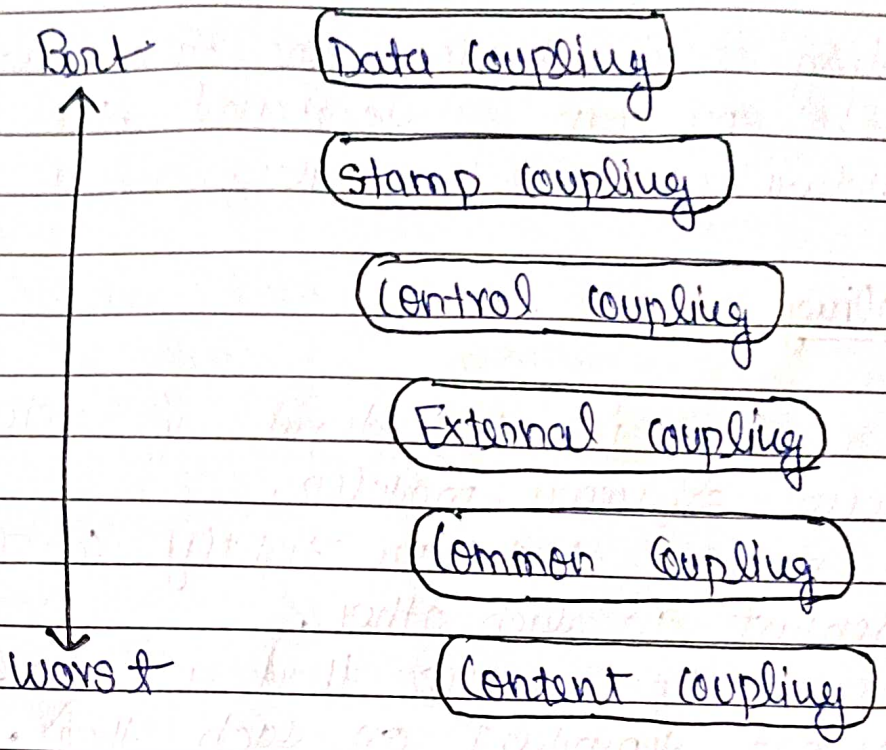
Loosely coupled Some dependencies (b)

Highly coupled Many dependencies (c)

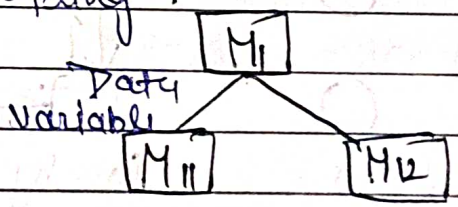
- A good design is the one that has low coupling.
- Coupling is measured by the number of relation between the modules.
- That is, the coupling increases as the number of calls between module increase.
- Thus it can be said that a design with high coupling will have more errors

Page No.:
Date : / /

Types of coupling:



- Data coupling: when data of one module is passed to another module, this is called data coupling.



- Stamp coupling: Two modules are stamp coupled if they communicate using composite data items such as structure, object, etc.

- When the module passes entire structure to another module, they are said to be stamp coupled.
For ex:- Passing structure variable in C or object in C++ language to a module.

Page No.
Date: / /

- Control coupling: If the modules communicate by passing control information, they are said to be control coupled.
- It can be bad if parameters indicate completely different behaviour and good if parameter allow factoring and reuse of functionality
  Ex: Sort function that takes comparison function as an argument

- External coupling: In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware.
  Ex: Protocol, device format, etc.

- Common coupling: The modules have shared data such as global data structures.
- The changes in global data mean tracking back to all modules which access that data to evaluate the effect of the change.
- So it has got disadvantage like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability

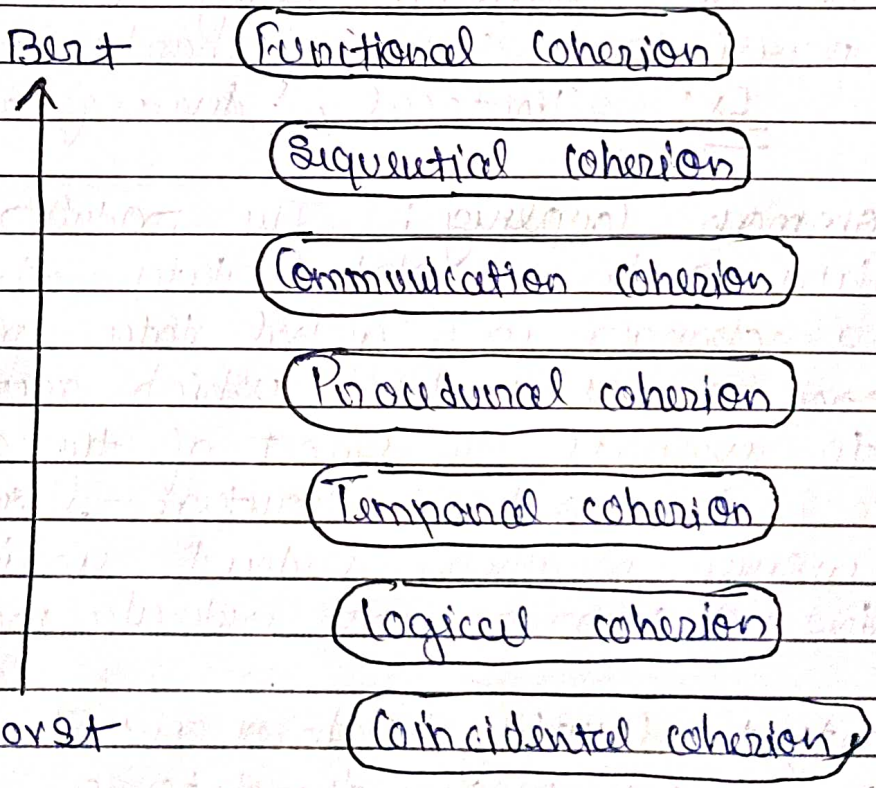
- Content coupling: In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module.
- This is the worst form of coupling and should be avoided.

Page No.:
Date : / /

⇒ Cohesion

- Cohesion is a measure of the degree to which the elements of the module are functionally related.
- It is the degree to which all elements directed towards performing a single task are contained in the component.

Types of cohesion



- Functional cohesion: Functional cohesion is said to exist if the different module elements cooperate to achieve a single function

Page No.
Date / /

- Sequential coherion: If the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.

- Communicational coherion :- If all tasks of the module refer to or update the same data structure e.g, the set of function defined on an array or a stack.

- Procedural coherion :- If the set of purpose of the module are all parts of a procedure in which particular sequence of step has to carried out for achieving goal. e.g the algorithm for decoding a message.

- Temporal coherion :- when a module includes functions that are associated by the fact that all the method must be executed in the same time.

- logical coherion : If all the element of the module perform a similar operation. For ex - Error handling, data input and data output etc.

- Coincidental coherion : If it perform a set of tasks that are associated with each other very loosely, if at all.

Page No.:
Date: / /

⇒ **Cohesion :-**

- Cohesion is the concept of intra-module.
- Cohesion represents the relationship within the module.
- Increasing cohesion is good for software.
- Cohesion represent the functional strength of modules
- Highly cohesive gives the best software.
- In cohesion, the module focuses on a single thing.
- Cohesion is created between the same module.
- Cohesion is of Six type.

⇒ **Coupling :-**

- Coupling is the concept of inter-module.
- Coupling represents the relationship between module.
- Increasing coupling is avoided for software.
- Coupling represents the independence among modules
- whereas loosely coupling gives the best software.
- In coupling modules are connected to the other modules
- Coupling is created between two different module.
- Coupling is of six type.

Page No.
Date / /

⇒ Design Strategies :— [AKTU – 21/22, 22/23]

⇒ Function Oriented Design :

- The basic abstractions, which are given to the user, are real world functions
- Function are grouped together by which a higher level function is obtained.
- Carried out using structured analysis and structured design i.e data flow diagram
- In this approach the state information is often represented is a centralized shared memory.
- It is a top down approach
- Begins by considering the use case diagrams and the scenarios.
- If function oriented design we decompose in function level.
- This approach is mainly used for computation sensitive application.
- It consist structure chart, flow chart, Pseudocode.

⇒ Object Oriented Design.

- The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.
- Function are grouped together on the basis of the data they operate since the classes are associated with their methods.
- Carried out using UML

- In this approach the state information is not represented in a centralized memory but is implemented or distributed among the objects of the system.
- It is a bottom up approach.
- Begins by identifying objects and classes.
- We decompose in class level.
- This approach is mainly used for evolving system which mimics a business or business case.
- It consist OOPS principle like class, encapsulation, Abstraction, object, inheritance, polymorphism.

⇒ Top - Down and Bottom - Up Design

- Top-Down Approach [AKTU - 2021 - 22]
- In this approach we focus on breaking up the problem into smaller parts
- Mainly used by structured programming language such as COBOL, Fortran, C, etc
- Each part is programmed seperately therefore contain redundancy.
- In this the communication is less among modules
- It is used in debugging, module documentation, etc
- In top-down approach, decomposition takes place.

Page No.
Date / /

- In this top function of system might be hard to identify.
- In this implimentation details may differ.

- Bottom up Approach

- In bottom up approach, we solve smaller problem and integrate it as whole and complete the solution.
- Mainly used by oop language such as C++, C# , python.
- Redundancy is minimized by using data encapsulation and data hiding.
- In this module must have communication
- It is basically used in testing.
- In bottom up approach composition takes place.
- In this sometimes we can not build a program from the piece we have started.
- This is not natural for people to assemble

Advantage of Top-down approach.
- More systematic
- Easy to understand and provide a modular architecture
- Should be used with small and medium size product
- Used mostly.

Disadvantage :-
- When a problem is very complex and very large then we can not understand the entire problem as a whole.

Page No.:
Date : / /

Advantage of bottom-up approach :-

- Should be used on large size projects.
- Easy to use by designers as we work in incremental fashion.

Disadvantage

- It has complex architecture very difficult to understand and manage.

Measurement :-

- A software metric is a measure of software characteristic which are measurable or countable.
- Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity and many other uses.
- Within the software development process, many metrics are that are all connected.
- Software metrics are similar to the four function of management : planning, Organization, control or improvement.

Need of Software Measurement :-

- Software is measured to :

→ Create the quality of the current product or process.

→ Anticipate future qualities of the product or processes.

Page No.
Date: / /

- Enhance the quality of a product or process.
- Regulate the state of the project in relation to budget and schedule.
- Enable the ongoing improvement of Software D.P.

Metrics:-

- A metric is a measurement of the level at which any input belongs to a system product or process

- Software metrics are a quantifiable or countable assessment of the attribute of a software product

=> There are 4 function related to software metrics

1. Planning
2. Organizing
3. Controlling
4. Improving.

=> Characteristics of software metrics

1. Quantitative: Metrics must possess a quantitative nature.
2. Understandable: Metric computation should be easily understood, and the method of computing metrics should be clearly defined.
3. Applicability: Metric should be applicable in the initial phases of the development of the software
4. Repeatable: When measured repeatedly, the metric value should be the same and consistent

Page No.:
Date : / /

5 Economical: The computation of metrics should be economical.

6. Language Independent: Metrics should not depend on any programming language

=> There are 8 types of software metrics:

- Product Metrics:

=> Product metrics are used to evaluate the state of the product, tracing risks and uncovering prospective problem areas.
=> The ability of team to control quality is evaluated

- Process Metrics:

=> Process metrics pay particular attention on enhancing the long term process of the team or organization.

- Project Metrics:

- The project matrix describes the project characteristic and execution process.
- Number of software developers
- Staffing pattern over the life cycle of software.
- Cost and schedule
- Productivity

Page No.
Date / /

Adv:

- Reduction in cost or budget.
- It helps to identify the particular area for improving.
- It helps to increase the productivity.
- Managing the workloads and teams.
- Reduction in overall time to produce the product.

Disadv:

- It is expensive and difficult to implement the metrics in some cases.
- Sometimes the quality of the product is not met with the expectation.
- It leads to measure the unwanted data which is wastage of time.

⇒ Halestead's Software Science.

⇒ Statement
A computer program is an implementation of an algorithm considered to be collection of token which can be classified as either operation or operand.

- All software science metrics can be defined in terms of these basic symbols. These symbols are called as token.

Page No.:
Date : / /

- The basic measures are
- $n1$ = count of unique operator
- $n2$ = count of unique operand
- $N1$ = count of total occurrences of operators
- $N2$ = count of total occurrence of operands.
- Size of the program can be expressed as $N = N_1 + N_2$.

- Program Volume (V)
- The unit of measurement of volume is the standard unit for size "bits".
- It is the actual size of a program if a uniform binary encoding for the vocabulary is used.

$$V = N * \log_2 n$$

- Program level (L)

The value of L ranges between zero and one with $L = 1$ representing a program written at the highest possible level (i.e with minimum size).

$$L = V^* / V.$$

- Program Difficulty (D)

The difficulty level or error-proneness (D) of the program is proportional to the number of the unique operator

in the program D = (n1/2) * (N2/n2)

- Program effort (E) :
- This is the product of program volume (V) and program difficulty.

$$E = V * D = V / L$$

- Language level : Language level implies that higher the level of a language, the less effort it takes to develop a program using that language.

→ language level is inversely proportional to the effort to develop a program.

$$L \propto \frac{1}{E}$$

- Potential Minimum Volume : The potential minimum volume V* is defined as the volume of the most succint program in which a problem can be coded

$$V^* = (2+n2^*) * \log_2(2+n2^*)$$

Adv :

- Predict error rate.
- Predict maintenance effort
- Simple to calculate

Page No.:
Date : / /

- Measure overall quality
- Used for any language

Disadvantage
- Depends on complete code
- Complexity increases as program level decreases.
- Difficult to compete

⇒ Function Point (FP) Based Measures

- Function point is used in the estimation of software development cost which is the most important potential use of function point data.

Function point may be used to compute the following importance metrics:

Productivity = FP/persons-months
Quality = Defects/FP
Cost = Rupees/FP

Organizations that use function point method develop criteria for determining whether a particular entry is low, average or high.

There are three types of transaction function
- External Inputs

Page No.
Date: / /

- External Outputs
- External Inqueries

Adv :-
- Size oriented metrics
- Language dependent
- Understood by the non technical user
- To estimate cost and resources required for software development

Disadv :-
- Manually Counting Process
- Difficult to Understand
- Requires Experience -

Types of FP attributes

⊙	Measurement Parameters	Examples
1.	Number of External Inputs (EI)	Input screen and table
2.	Number of External Output (EO)	Output Screen and report
3.	Number of External inquiries (EQ)	Prompts and interrupt
4.	Number of internal files (ILF)	Databases and directories.
5.	Number of external interfaces (EIF)	shared databases and shared inquiries

Page No.:
Date : / /

⇒ Cyclomatic Complexity

- Cyclomatic complexity is a software metric that measures the complexity of a program's control flow by counting the number of linearly independent paths through the source code.

Need :

- Assess code maintainability and readability
- Identify potential error and areas of high risk.
- Determine testing effort required
- Aid in refactoring effort to simplify code.

objective :

- Quantify the complexity of a program or function.
- Facilitate better decision-making in software development.
- Improve overall code quality and understandability.

Process :

- Identify the program decision points (such as if, while, and for statements)

Page No.
Date / /

- Calculate the number of linearly independent paths by counting the decision point and adding one.
- Interpret the resulting value:

→ Low value: lower complexity, easier to maintain and test.

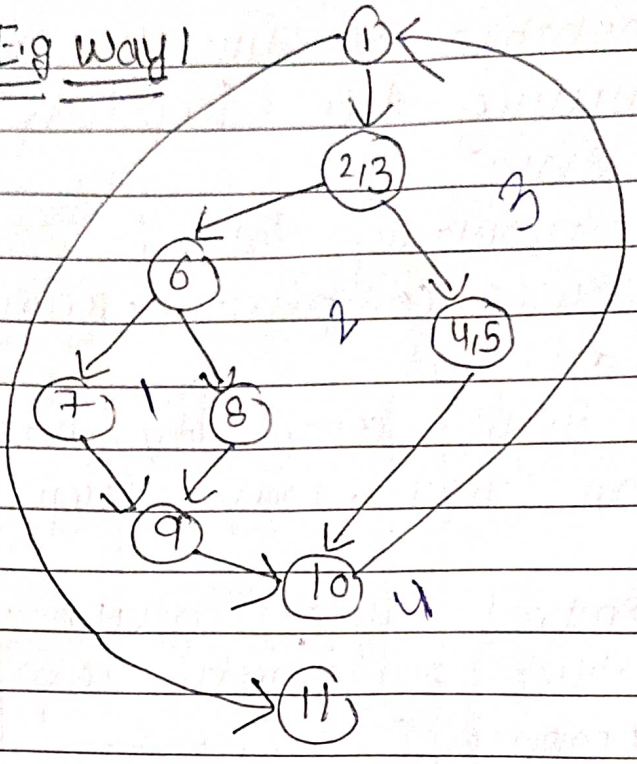
→ High value: Higher complexity, harder to maintain and test, may require refactoring

⇒ Given a control flow graph G of a program, the cyclomatic complexity V(G) can be computed as:

- V(G) = E - N + 2, where E is the no. of edges N is the total number of nodes
- V(G) = V(G) = Total number of bounded area + 1 Any region enclosed by nodes and edges can be called as a bounded area
- The cyclomatic complexity of a program can also be easily computed by counting the number of decision statement of the program.

Page No.:
Date : / /

E.g Way 1



Number of edges = 11
Number of vertices = 9

Cyclomatic complexity = 11 - 9 + 2
⇒ 4.

Way 2 By finding Bounded region and outing region.

① Number of Bounded region = 3
" " outer region = 1
Cyclomatic complexity = 3 + 1
= 4

Or

Cyclomatic complexity = Number of Bounded region + 1
⇒ 3 + 1
⇒ 4

[AKTU-21/22]

Q Draw the software design framework and discuss the elments of design model.

Sol Elmnt of a system

1. Architecture: This is the conceptual model that define the structure, behavior, and views of a system.

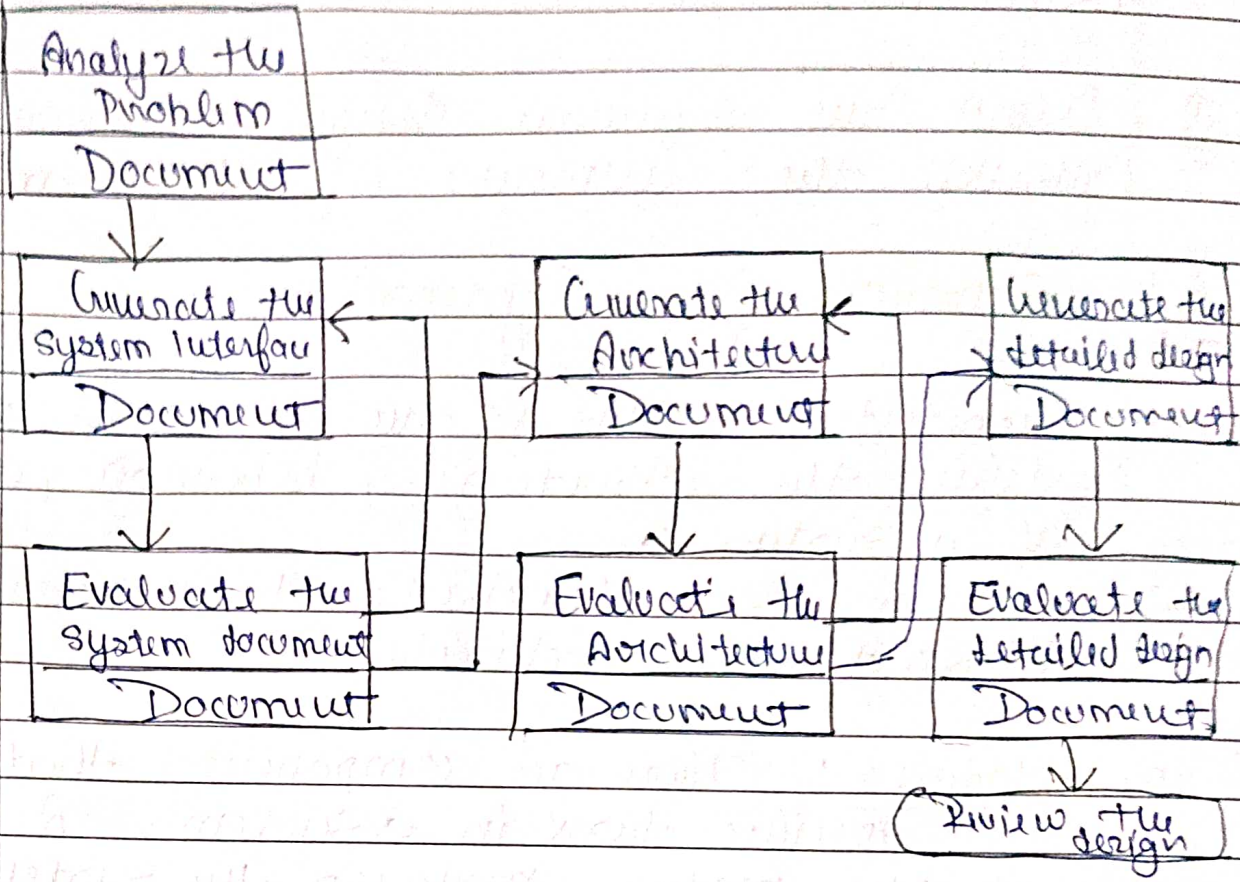
→ We can use flowcharts to represent and illustrate the architecture.

2. Modules: There are components that handle one specific task in a system. A combination of the module make up the system

3. Components: This provides a particular function or group of related functions. They are made up of modules.

4. Interfaces: This is the shared boundary across which the component of a system exchange information and relate.

5. Data: This is the management of the information and data flow



Unit-3 is Completed

Subscribe

Check Description for Notes