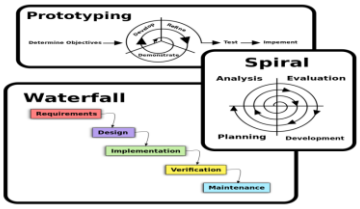
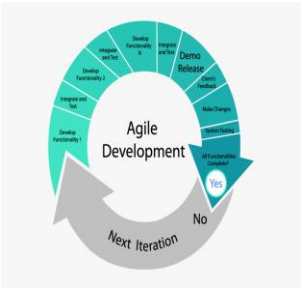
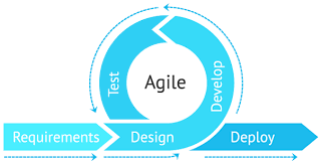


Software Engineering Design Concept



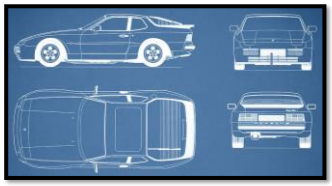


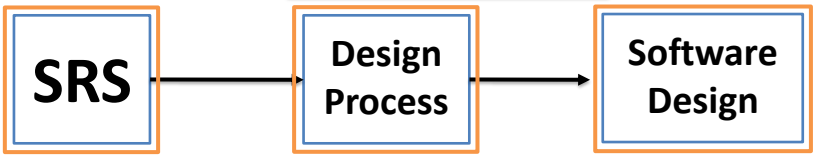
Outline

- Abstraction
- Architecture
- Aspects
- Cohesion
- Coupling,
- Data Design
- Design Process
- Functional Independence
- Good Design
- Information Hiding.

What is Design

- A **meaningful representation** of something to be built
- It's a **process** by which **requirements** are **translated** into **blueprint** for constructing a software
- **Blueprint** gives us the **holistic view** (entire view) of a **software**





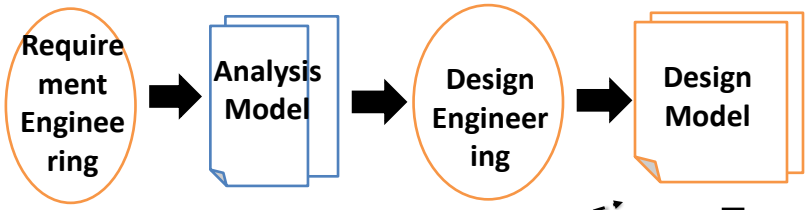
Problem Domain

Solution Domain

3

Software Design Process?

- Software **design** is the **most creative part** of the development **process**



Blue Print for Constructing Software

Process involves **Diversification** & **Convergence**

4

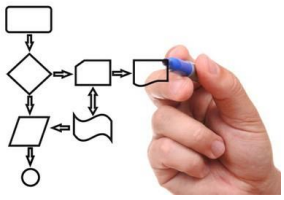
Software design work products

- For a design to be easily implemented in a conventional programming language, the following items must be designed during the design phase.
- **Different modules** required to implement the design solution.
- **Control relationship among** the identified **modules**. The relationship is also known as the call relationship or invocation relationship among modules.
- **Interface among** different **modules**. The interface among different modules identifies the exact data items exchanged among the modules.
- **Algorithms** required **to implement** each individual **module**.
- **Data structures of** the individual **modules**.

5

Characteristics of good Design

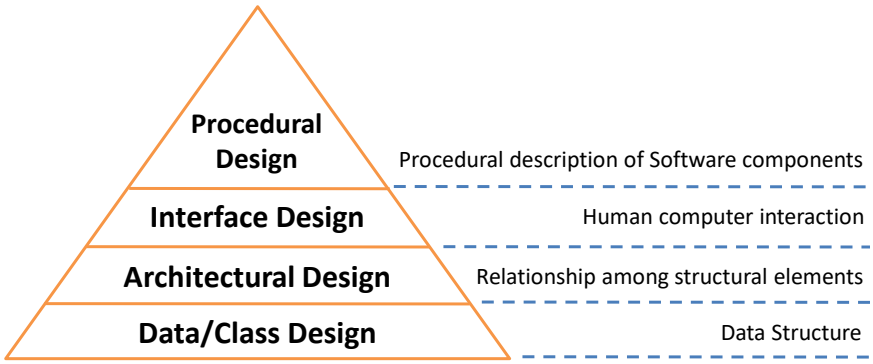
- The design must **implement all explicit requirements** available in requirement model.
- The design must **accommodate all implicit requirements** given by stakeholders.
- The design must be **readable** & **understandable**
- The good design should **provide complete picture of the software**, addressing the **data**, **functional** and **behavioral** domains.



6

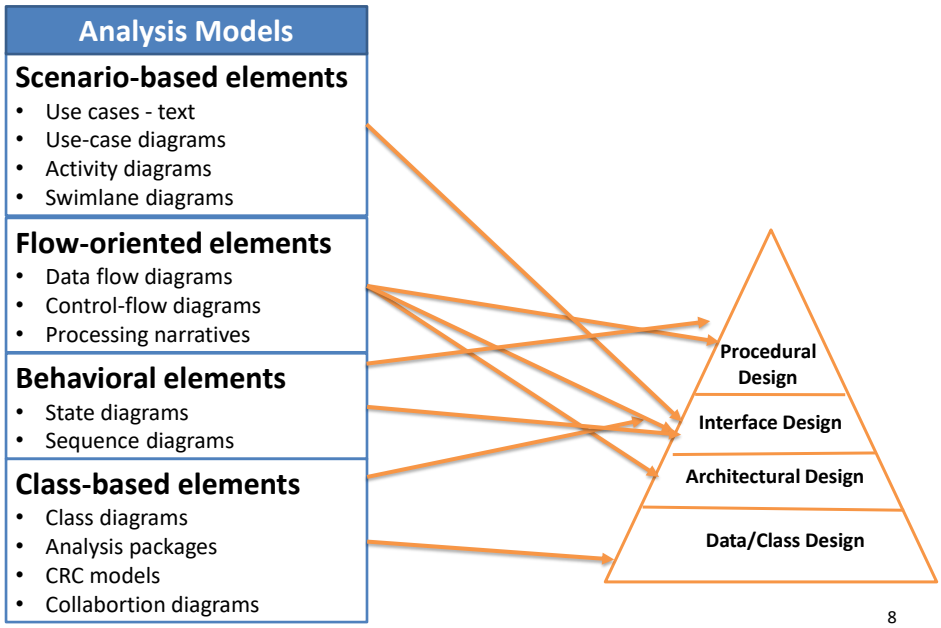
Design Models

- It is **creative** activity. It is most **critical activity** during system development
- Has great **impact** on **testing** and **maintenance**
- Design document forms **reference for later phases**



7

Design Models



8

Design Models

Data Design

It **transforms class models** into **design class** realization and **prepares data structure** (**data design**) required to implement the software.

Architectural Design

It **defines** the **relationship between** major **structural elements** of the software

9

Design Models

Interface Design

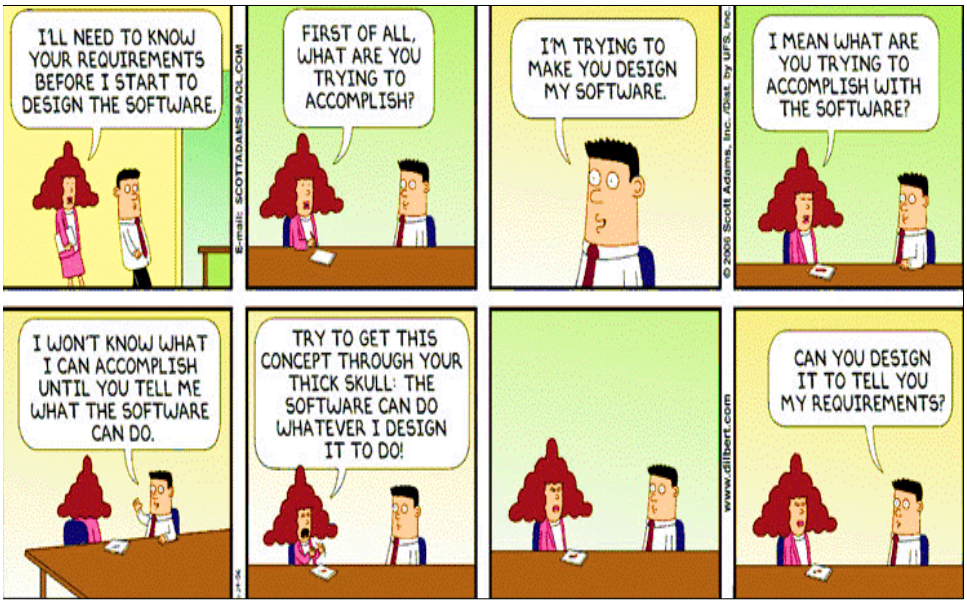
It defines **how software communicates** with **systems** & with **humans**. An interface implies flow of information & behavior.

Procedural Design

It **transforms structural elements** of software into **procedural description** of software components

10

Gentle overview of – F U R P S



11

Quality attributes of software design (FURPS)

F **Functionality**

assessed by **feature set** and **capabilities of** the **program**, **generality** of the **functions** & **security** of overall **system**

U **Usability**

assessed by considering **human factors**, overall **aesthetics**, **consistency** & **documentations**

R **Reliability**

assessed by measuring **frequency** & **severity** of **failures**, **accuracy** of **outputs**, **mean-time-of-failure (MTTF)**, **ability** to **recover** from **errors**

P **Performance**

measured by **processing speed**, **response time**, **resource consumption**, **throughput** and **efficiency**

S **Supportability**

Ability to extend program, adaptability, serviceability, testability, compatibility

Design Concepts

- The beginning of **wisdom** for a **software engineer** is to **recognize** the **difference** between **getting program to work** and **getting it right.**
- **Fundamental software design** concepts provide the necessary framework for **"getting it right."**
- Each design concept helps to answer the following questions
  1. What criteria can be used to partition software into individual components?
  2. How is function or data structure detail separated from a conceptual representation of the software?
  3. What uniform criteria define the technical quality of a software design?

13

Important Software Design Concepts

Abstraction	Architecture	Pattern
Separation of Concern	Modularity	Information Hiding
Refactoring	Refinement	Aspects
Functional Independence		

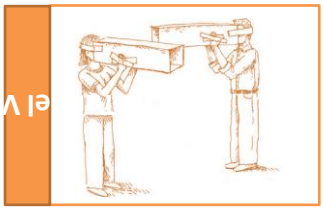
Abstraction

- At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment. At lower levels of abstraction, a more detailed description of the solution is provided. Finally, at the lowest level of abstraction, the solution is stated in a manner that can be directly implemented.
- Creation of procedural and data abstractions are done.
- A procedural abstraction refers to a sequence of instructions that have a specific and limited function. A data abstraction is a named collection of data that describes a data object.

14

Design Principles

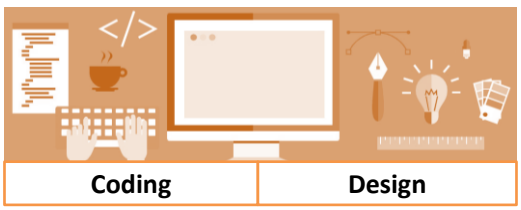
- Design process should **not suffer** from **"tunnel vision"**
- Design should be **traceable** to the **analysis model**
- Design should **not reinvent** the **wheel**
- Design should **"minimize the intellectual distance"** between the **software** and **the real world** problem
- Design should exhibit (**present**) **uniformity** and **integration**
- Design should be **structured to accommodate change**



15

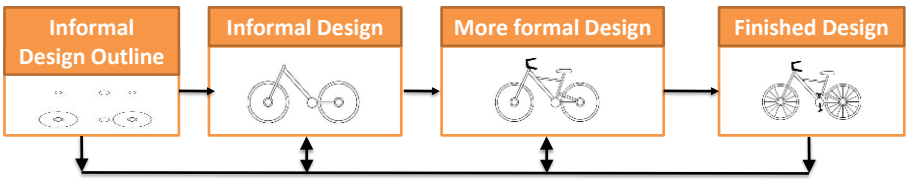
Design Principles

- Design should be **structured** to **degrade gently**, even when abnormal data, events, or operating conditions are encountered
- Design is not coding, coding is not design
- **Design** should be **assessed for quality** as it is **being created**, not after the fact
- Design should be **reviewed to minimize conceptual** (semantic) errors

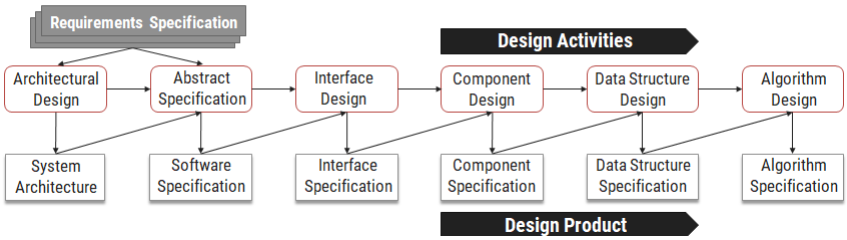


16

Design Process Rough View



Design Process

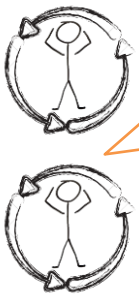


17

Cohesion & Coupling

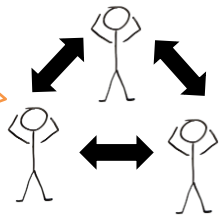
A **good software design** implies **clean decomposition** of the **problem into modules**, and the **neat arrangement** of these **modules** in a **hierarchy**.

The primary **characteristics** of **neat module decomposition** are **high cohesion** and **low coupling**.



A **cohesive module** performs a single task, requiring little interaction with other componen ts.

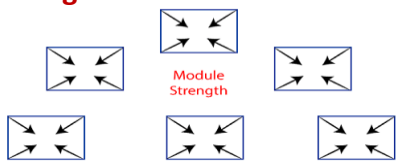
A **Coupling** is an indication of the relative interdependence among modules.



18

Cohesion

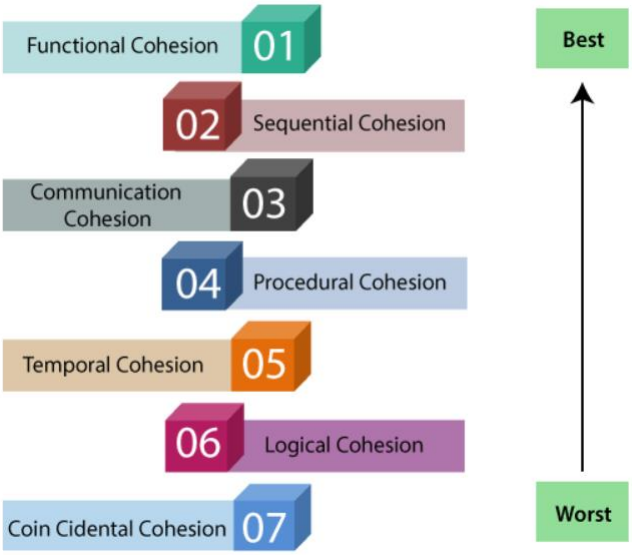
- Cohesion is an **indication** of the **relative functional strength** of a module.
- A **cohesive module** performs a **single task**, **requiring little interaction** with other components.
- Stated simply, a **cohesive module** should (ideally) **do just one thing**.
- A module having **high cohesion** and **low coupling** is said to be **functionally independent** of other modules.
- By the term functional independence, we mean that a **cohesive module performs a single task or function**.



Cohesion= Strength of relations within Modules

19

Classification of Cohesion



20

Classification of Cohesion

- **Coincidental Cohesion**
  - A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely.
  - In this case, the module contains a random collection of functions.
  - It is likely that the functions have been put in the module out of pure coincidence without any thought or design.
  - For Ex., in a transaction processing system (TPS), the get-input, print-error, and summarize-members functions are grouped into one module
  - At the **outer layer**, components service **user interface** operations.

21

Classification of Cohesion

- **Logical cohesion**
  - A module is said to be logically cohesive, if all elements of the module perform similar operations.
  - For Ex., error handling, data input, data output, etc.
  - An example of logical cohesion is the case where a set of print functions generating different output reports are arranged into a single module In this case, the module contains a random collection of functions.
- **Temporal cohesion**
  - When a module contains functions that are related by the fact that all the functions must be executed in the same time span.
  - For Ex., the set of functions responsible for initialization, start-up, shutdown of some process, etc.

22

Classification of Cohesion

- **Procedural cohesion**
  - If the set of functions of the module are all part of a procedure (algorithm) in which certain sequence of steps have to be carried out for achieving an objective
  - For Ex., the algorithm for decoding a message.
- **Communicational cohesion**
  - If **all functions** of the module **refer** to the **same data structure**
  - For Ex., the set of functions defined on an array or a stack.

23

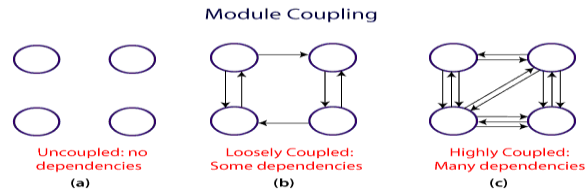
Classification of Cohesion

- **Sequential cohesion**
  - If the elements of a module form the parts of sequence, where the output from one element of the sequence is input to the next.
  - For Ex., In a Transaction Processing System, the get-input, validate-input, sort-input functions are grouped into one module.
- **Functional cohesion**
  - If different elements of a module cooperate to achieve a single function.
  - For Ex., A module containing all the functions required to manage employees' pay-roll exhibits functional cohesion.

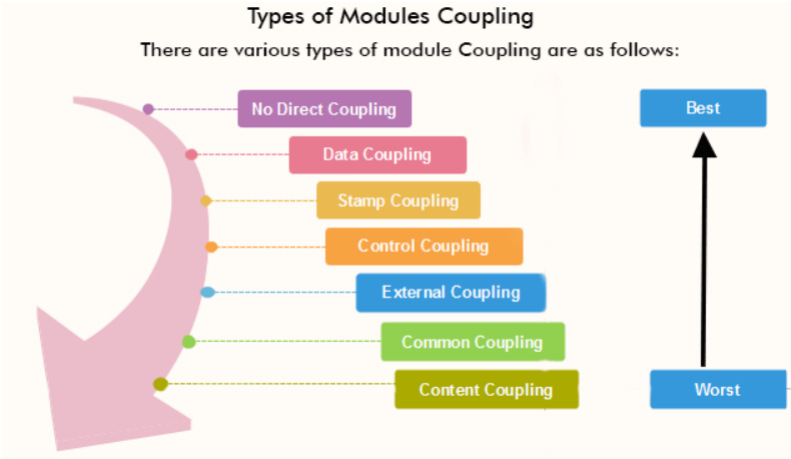
24

Coupling

- **Coupling** between two modules is a **measure of the degree** of **interdependence** or interaction **between** the **two modules**.
- A module having high cohesion and low coupling is said to be functionally independent of other modules.
- If **two modules interchange large amounts of data**, then they are **highly interdependent**.
- The degree of coupling between two modules depends on their interface complexity.
- The interface complexity is basically determined by the number of types of parameters that are interchanged while invoking the functions of the module.

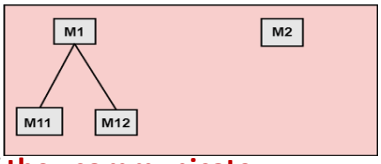


Classification of Coupling

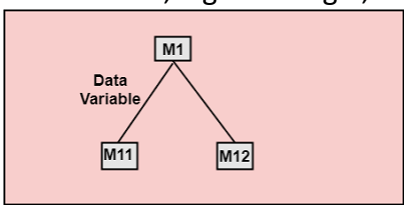


Classification of Coupling

- **No direct coupling**
  - There is no direct coupling between M1 and M2
  - In this case, modules are subordinates to different modules. Therefore, no direct coupling.



- **Data coupling**
  - Two modules are data coupled, if **they communicate through a parameter**.
  - An example is an elementary (primal) data item passed as a parameter between two modules, e.g. an integer, a float, a character, etc.



27

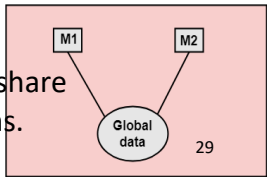
Classification of Coupling

- **Stamp coupling**
  - This is a special case (or extension) of data coupling
  - Two modules (``A'' and ``B'') exhibit stamp coupling if **one passes** directly to the other a **composite data item** - such as a record (or structure), array, or (pointer to) a list or tree.
  - This occurs when **ClassB** is **declared as a type** for an argument of an **operation of ClassA**
- **Control coupling**
  - If **data** from **one** module **is used to direct the order of instruction**s execution **in another**.
  - An example of control coupling is a flag set in one module and tested in another module

28

Classification of Coupling

- **Content coupling**
  - Content coupling occurs when **one component secretly modifies data** that is **internal to another component.**
  - This violets information hiding – a basic design concept.
  - Content coupling exists between two modules, if they share code e.g., a branch from one module into another module.
- **External coupling**
  - External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.
- **Common coupling**
  - Two modules are common coupled if they share information through some global data items.



Difference between Cohesion and Coupling

Cohesion	Coupling
Cohesion is the concept of intra-module.	Coupling is the concept of inter-module.
Cohesion represents the relationship within a module.	Coupling represents the relationships between modules.
Increasing cohesion is good for software.	Increasing coupling is avoided for software.
Cohesion represents the functional strength of modules.	Coupling represents the independence among modules.
Highly cohesive gives the best software.	Whereas loosely coupling gives the best software.
In cohesion, the module focuses on a single thing.	In coupling, modules are connected to the other modules.
Cohesion is created between the same module.	Coupling is created between two different modules.

Aspect

- As requirements analysis occurs, a set of “concerns” is uncovered. These concerns “include requirements, use cases, features, data structures, quality-of-service issues, variants, intellectual property boundaries(Software intellectual property, also known as software IP, is **a computer code or program that is protected by law against copying, theft, or other use that is not permitted by the owner**. Software IP belongs to the company that either created or purchased the rights to that code or software.), collaborations, patterns and contracts”.
- Ideally, a requirements model can be organized in a way that allows you to isolate each concern (requirement) so that it can be considered independently.

31

Aspect

- As design begins, requirements are refined into a modular design representation.
- Consider two requirements, A and B. Requirement A crosscuts requirement B “ if a software decomposition [refinement] has been chosen in which B cannot be satisfied without taking A into account”

32

Information hiding

- The concept of modularity leads you to a fundamental question: “How do I decompose a software solution to obtain the best set of modules?”
- The principle of information hiding suggests that modules be “characterized by design decisions that (each) hides from all others.”
- In other words, modules should be specified and designed so that information (algorithms and data) contained within a module is inaccessible to other modules that have no need for such information.
- Because most data and procedural detail are hidden from other parts of the software, inadvertent errors introduced during modification are less likely to propagate to other locations within the software.

33

Information hiding

- The use of information hiding as a design criterion for modular systems provides the greatest benefits when modifications are required during testing and later during software maintenance.

34

Functional independence

- Functional independence is achieved by developing modules with “singleminded” function.
- Stated another way, you should design software so that each module addresses a specific subset of requirements and has a simple interface when viewed from other parts of the program structure.
- Software with effective modularity, that is, independent modules, is easier to develop because function can be compartmentalized and interfaces are simplified.
- Independent modules are easier to maintain (and test) because secondary effects caused by design or code modification are limited, error propagation is reduced, and reusable modules are possible.

35

Functional independence

- To summarize, functional independence is a key to good design, and design is the key to software quality.
- Independence is assessed using two qualitative criteria: cohesion and coupling.
- Cohesion is an indication of the relative functional strength of a module.
- Coupling is an indication of the relative interdependence among modules.

36


THANKS