

DESIGN & ARCHITECTURAL ENGINEERING



Design Modeling in Software Engineering

Component level design —

User Interfaces design —

Architectural design —

Data design —



Introduction to Design Modelling in Software Engineering

- Design modeling in software engineering represents the features of the software that helps engineer to develop it effectively, the architecture, the user interface, and the component level detail.
- Design modeling provides a variety of different views of the system like architecture plan for home or building.
- Different methods like data-driven, pattern-driven, or object-oriented methods are used for constructing the design model.
- All these methods use set of design principles for designing a model.

Working of Design Modelling in Software Engineering

- Designing a model is an important phase and is a multi-process that represent the data structure, program structure, interface characteristic, and procedural details.
- It is mainly classified into four categories –
 - **Data design,**
 - **Architectural design,**
 - **Interface design, and**
 - **Component-level design.**

Working of Design Modelling in Software Engineering

Data design:

- It represents the data objects and their interrelationship in an entity-relationship diagram.
- Entity-relationship consists of information required for each entity or data objects as well as it shows the relationship between these objects.
- It shows the structure of the data in terms of the tables. It shows three type of relationship – One to one, one to many, and many to many.
- In one to one relation, one entity is connected to another entity. In one many relation, one Entity is connected to more than one entity. un many to many relations one entity is connected to more than one entity as well as other entity also connected with first entity using more than one entity.

Working of Design Modelling in Software Engineering

Architectural design:

- It defines the relationship between major structural elements of the software.
- It is about decomposing the system into interacting components.
- It is expressed as a block diagram defining an overview of the system structure – features of the components and how these components communicate with each other to share data.
- It defines the structure and properties of the component that are involved in the system and also the inter-relationship among these components.

Working of Design Modelling in Software Engineering

User Interfaces design:

- It represents how the Software communicates with the user i.e. the behavior of the system.
- It refers to the product where user interact with controls or displays of the product.
- For example, Military, vehicles, aircraft, audio equipment, computer peripherals are the areas where user interface design is implemented.
- UI design becomes efficient only after performing usability testing.
- This is done to test what works and what does not work as expected.
- Only after making the repair, the product is said to have an optimized interface.

Working of Design Modelling in Software Engineering

Component level design:

- It transforms the structural elements of the software architecture into a procedural description of software components.
- It is a perfect way to share a large amount of data.
- Components need not be concerned with how data is managed at a centralized level i.e. components need not worry about issues like backup and security of the data.

Principles of Design Model

Design must be traceable to the analysis model:

- Analysis model represents the information, functions, and behavior of the system.
- Design model translates all these things into architecture – a set of subsystems that implement major functions and a set of component level design that are the realization of Analysis classes.
- This implies that design model must be traceable to the analysis model.

Principles of Design Model

Always consider architecture of the system to be built:

- Software architecture is the skeleton of the system to be built.
- It affects interfaces, data structures, behavior, program control flow, the manner in which testing is conducted, maintainability of the resultant system, and much more.

Principles of Design Model

Focus on the design of the data:

- Data design encompasses the manner in which the data objects are realized within the design.
- It helps to simplify the program flow, makes the design and implementation of the software components easier, and makes overall processing more efficient.

Principles of Design Model

User interfaces should consider the user first:

- The user interface is the main thing of any software.
- No matter how good its internal functions are or how well designed its architecture is but if the user interface is poor and end-users don't feel ease to handle the software then it leads to the opinion that the software is bad.

Principles of Design Model

Components should be loosely coupled:

- Coupling of different components into one is done in many ways like via a component interface, by messaging, or through global data.
- As the level of coupling increases, error propagation also increases, and overall maintainability of the software decreases.
- Therefore, component coupling should be kept as low as possible.

Principles of Design Model

Interfaces both user and internal must be designed:

- The data flow between components decides the processing efficiency, error flow, and design simplicity.
- A well-designed interface makes integration easier and tester can validate the component functions more easily.

Component level design should exhibit Functional independence:

- It means that functions delivered by component should be cohesive i.e. it should focus on one and only one function or sub-function.

We have discussed the basics of design modeling in software engineering along with its principles. We have also discussed its working and some other areas.

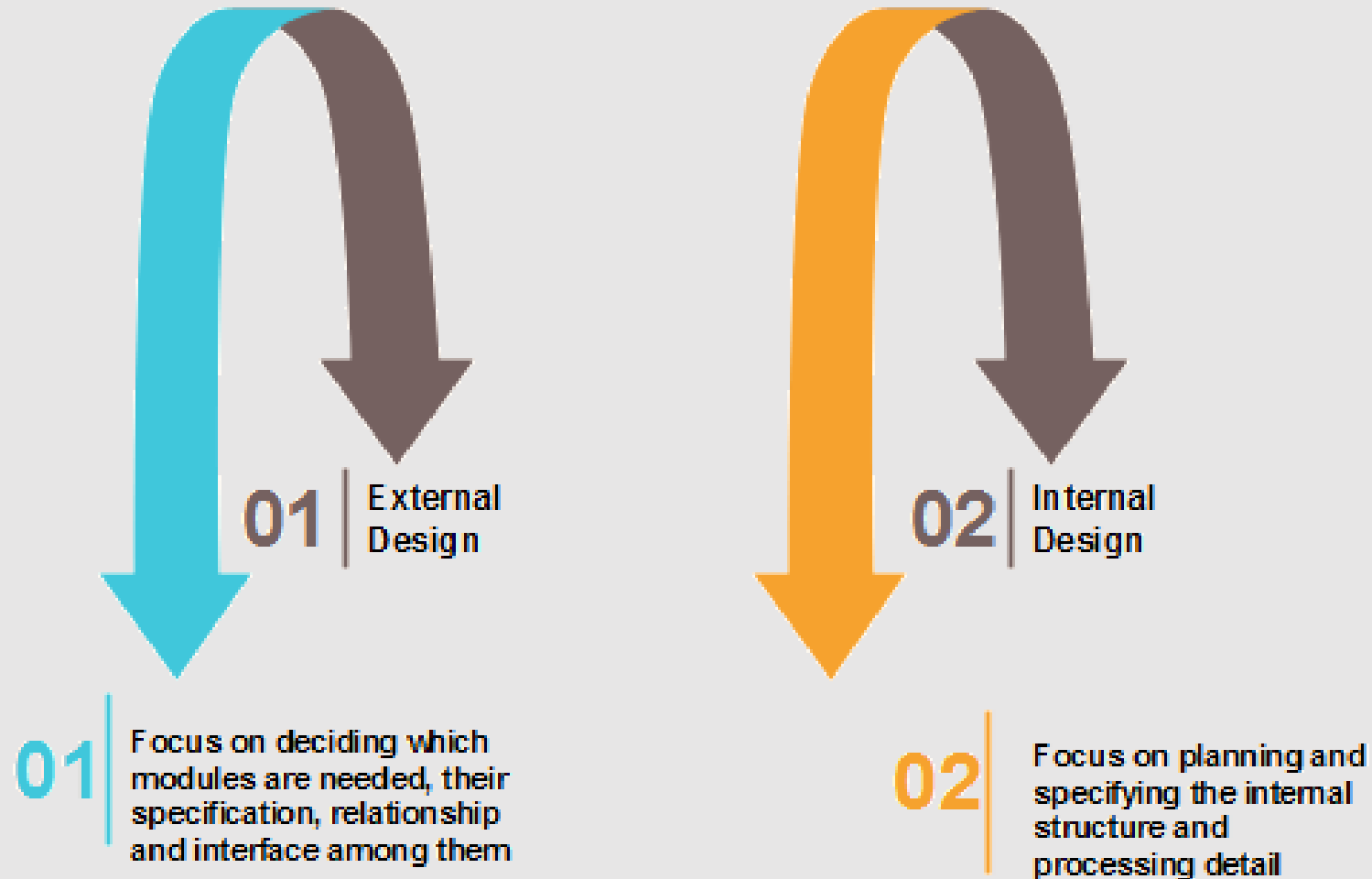
Software Design

- Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- It deals with representing the client's requirement, as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable using programming language.
- The software design phase is the first step in **SDLC (Software Design Life Cycle)**, which moves the concentration from the problem domain to the solution domain.
- In software design, we consider the system to be a set of components or modules with clearly defined behaviors & boundaries.

Software Design

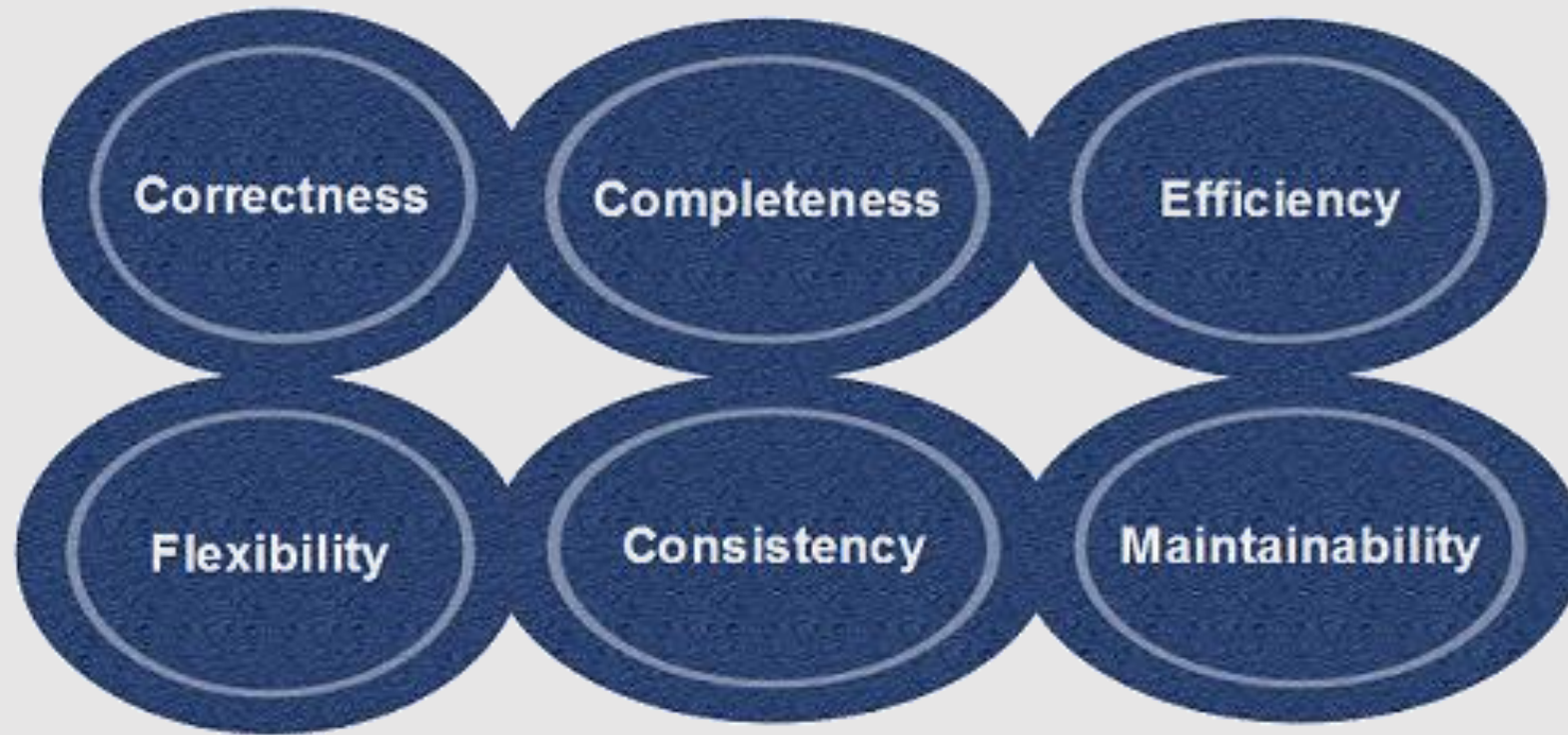
Software Design Levels

Software design process have two levels:



Objectives of Software Design

Following are the purposes of Software design:



Objectives of Software Design

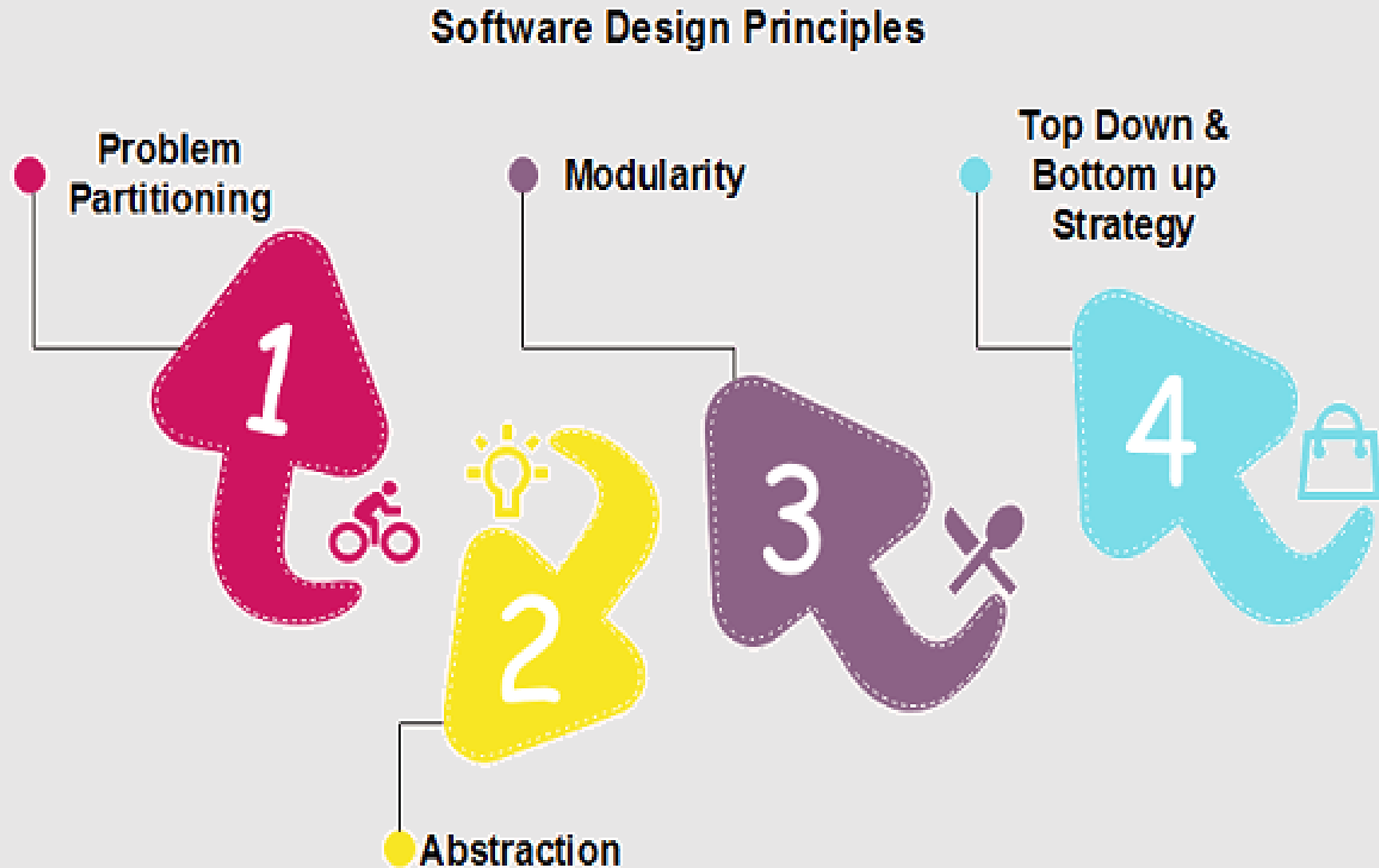
Objectives of Software Design

- **Correctness:** Software design should be correct as per requirement.
- **Completeness:** The design should have all components like data structures, modules, and external interfaces, 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.

Software Design Principles

- Software design principles are concerned with providing means to handle the complexity of the design process effectively.
- Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.

Software Design Principles



Software Design Principles – Problem Partitioning

- Problem Partitioning
- For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.
- For software design, the goal is to divide the problem into manageable pieces.

Software Design Principles - Problem Partitioning

- Benefits of Problem Partitioning
 1. Software is easy to understand
 2. Software becomes simple
 3. Software is easy to test
 4. Software is easy to modify
 5. Software is easy to maintain
 6. Software is easy to expand
- These pieces cannot be entirely independent of each other as they together form the system. They have to cooperate and communicate to solve the problem. This communication adds complexity.

Software Design Principles - Abstraction

- An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.
- Here, there are two common abstraction mechanisms
- Functional Abstraction
- Data Abstraction

Software Design Principles - Abstraction

Functional Abstraction

- A module is specified by the method it performs.
- The details of the algorithm to accomplish the functions are not visible to the user of the function.
- Functional abstraction forms the basis for **Function oriented design approaches**.

Data Abstraction

- Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for **Object Oriented design approaches**.

Software Design Principles - Modularity

- Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software.
- It is the only property that allows a program to be intellectually manageable.
- Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

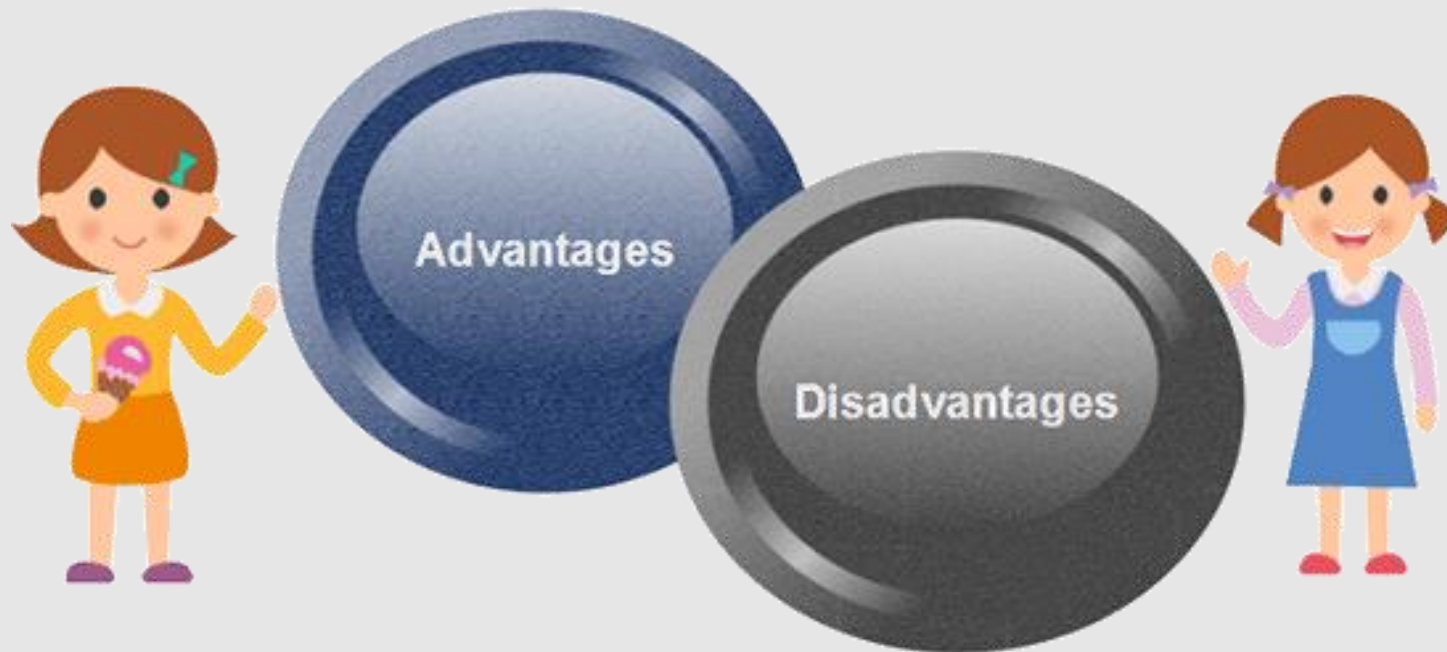
The desirable properties of a modular system are:

- Each module is a well-defined system that can be used with other applications.
- Each module has single specified objectives.
- Modules can be separately compiled and saved in the library.
- Modules should be easier to use than to build.
- Modules are simpler from outside than inside.

Software Design Principles - Modularity

Advantages and Disadvantages of Modularity

- In this topic, we will discuss various advantage and disadvantage of Modularity.



Advantages of Modularity

There are several advantages of Modularity

- It allows large programs to be written by several or different people
- It encourages the creation of commonly used routines to be placed in the library and used by other programs.
- It simplifies the overlay procedure of loading a large program into main storage.
- It provides more checkpoints to measure progress.
- It provides a framework for complete testing, more accessible to test
- It produced the well designed and more readable program.

Disadvantages of Modularity

There are several disadvantages of Modularity

- Execution time maybe, but not certainly, longer
- Storage size perhaps, but is not certainly, increased
- Compilation and loading time may be longer
- Inter-module communication problems may be increased
- More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done

Modular Design

Modular design reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system.

We discuss a different section of modular design in detail in this section:

1. Functional Independence:

- Functional independence is achieved by developing functions that perform only one kind of task and do not excessively interact with other modules.
- Independence is important because it makes implementation more accessible and faster.
- The independent modules are easier to maintain, test, and reduce error propagation and can be reused in other programs as well.
- Thus, functional independence is a good design feature which ensures software quality.

Modular Design

It is measured using two criteria:

Cohesion: It measures the relative function strength of a module.

Coupling: It measures the relative interdependence among modules.

2. Information hiding:

- The fundamental of Information hiding suggests that modules can be characterized by the design decisions that protect from the others, i.e., In other words, modules should be specified that data include within a module is inaccessible to other modules that do not need for such information.
- The use of information hiding as design criteria for modular system provides the most significant benefits when modifications are required during testing's and later during software maintenance.
- This is because as most data and procedures are hidden from other parts of the software, inadvertent errors introduced during modifications are less likely to propagate to different locations within the software.

Strategy of Design

A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.

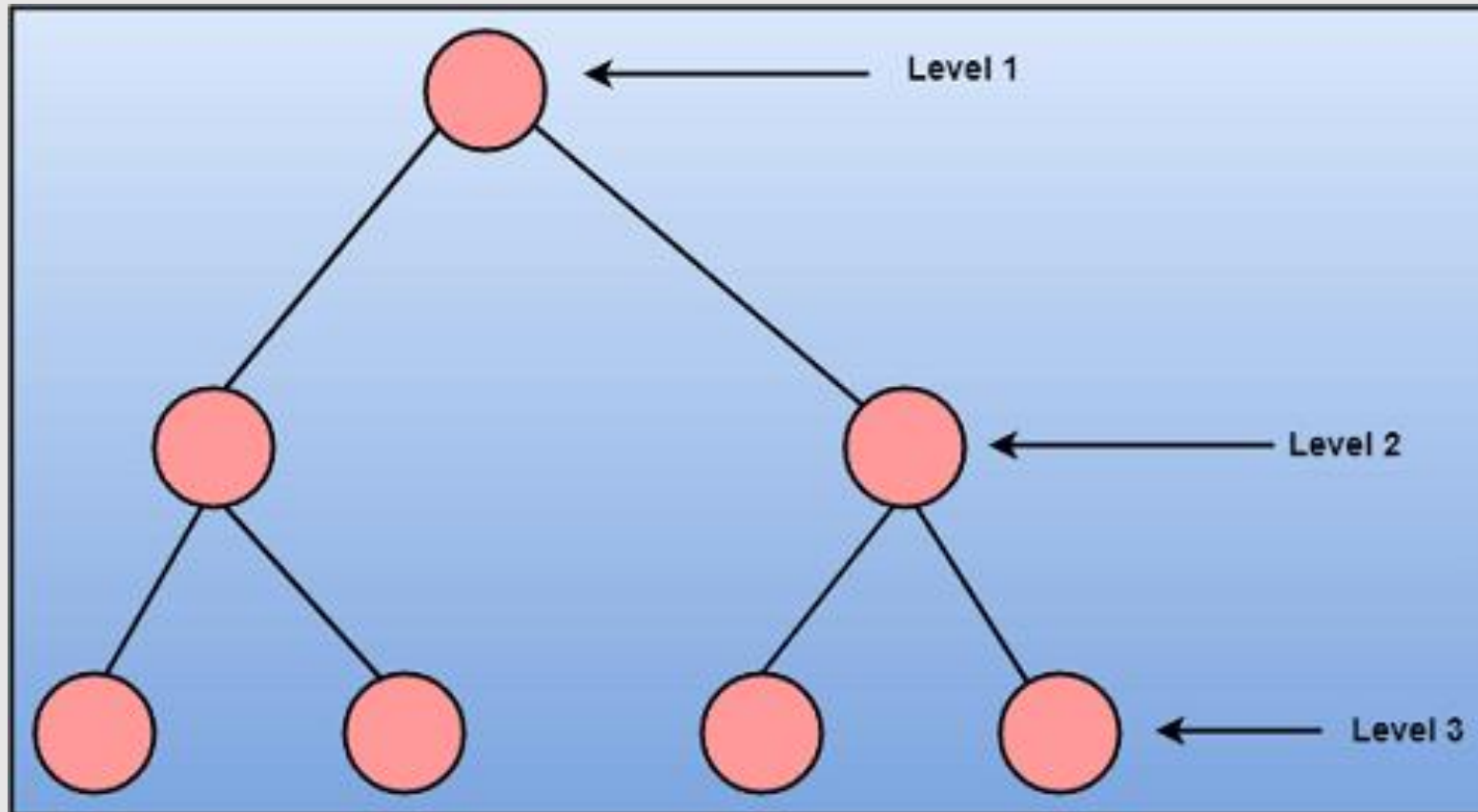
To design a system, there are two possible approaches:

Top-down Approach

Bottom-up Approach

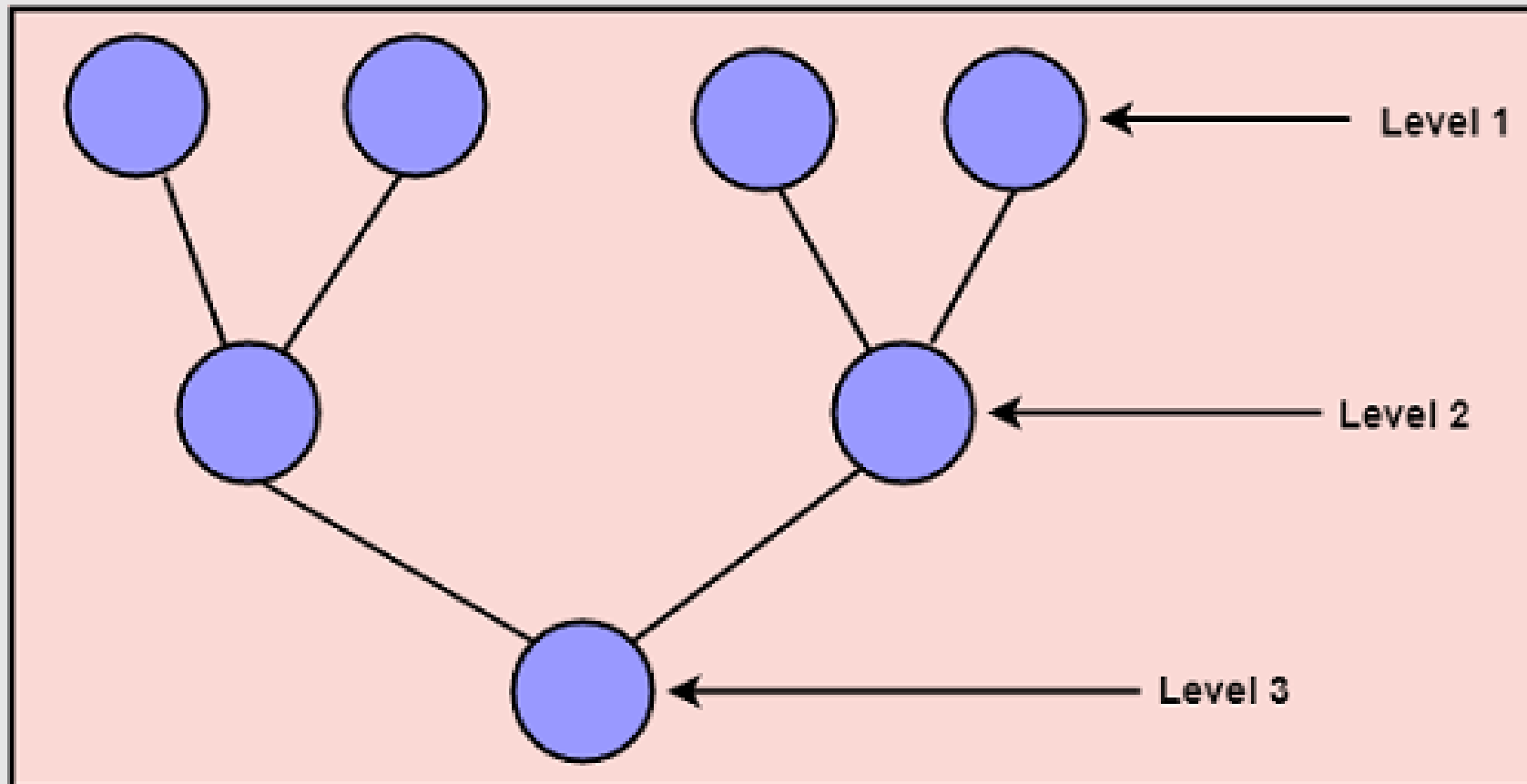
Strategy of Design

1. **Top-down Approach:** This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.



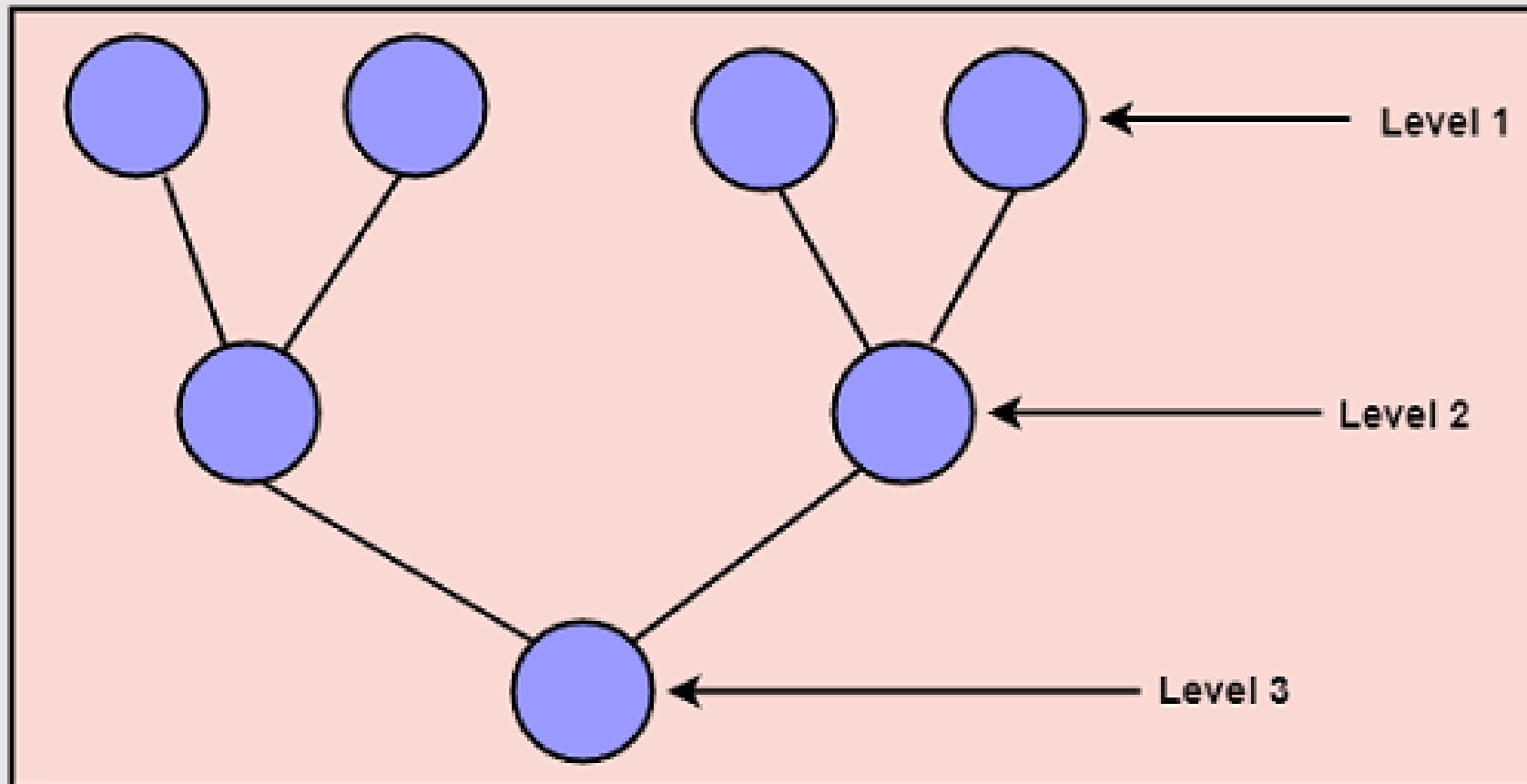
Strategy of Design

2. **Bottom-up Approach:** A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.



Strategy of Design

2. **Bottom-up Approach:** A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.



Coupling & Cohesion

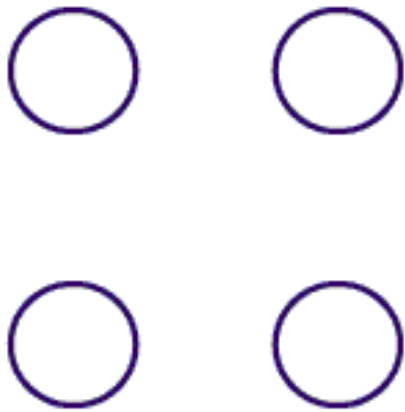
Module Coupling

In software engineering, 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.

The various types of coupling techniques are shown in fig:

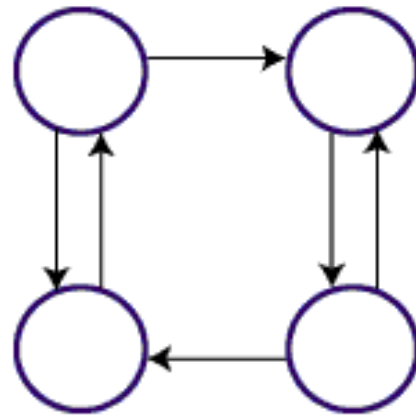
Coupling & Cohesion

Module Coupling



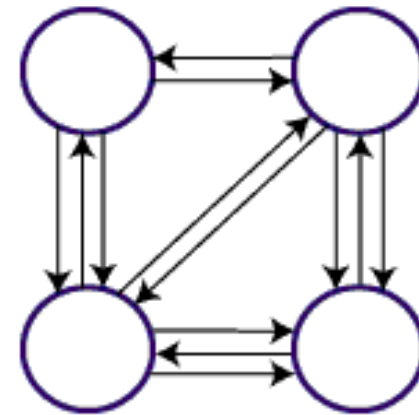
Uncoupled: no dependencies

(a)



Loosely Coupled: Some dependencies

(b)



Highly Coupled: Many dependencies

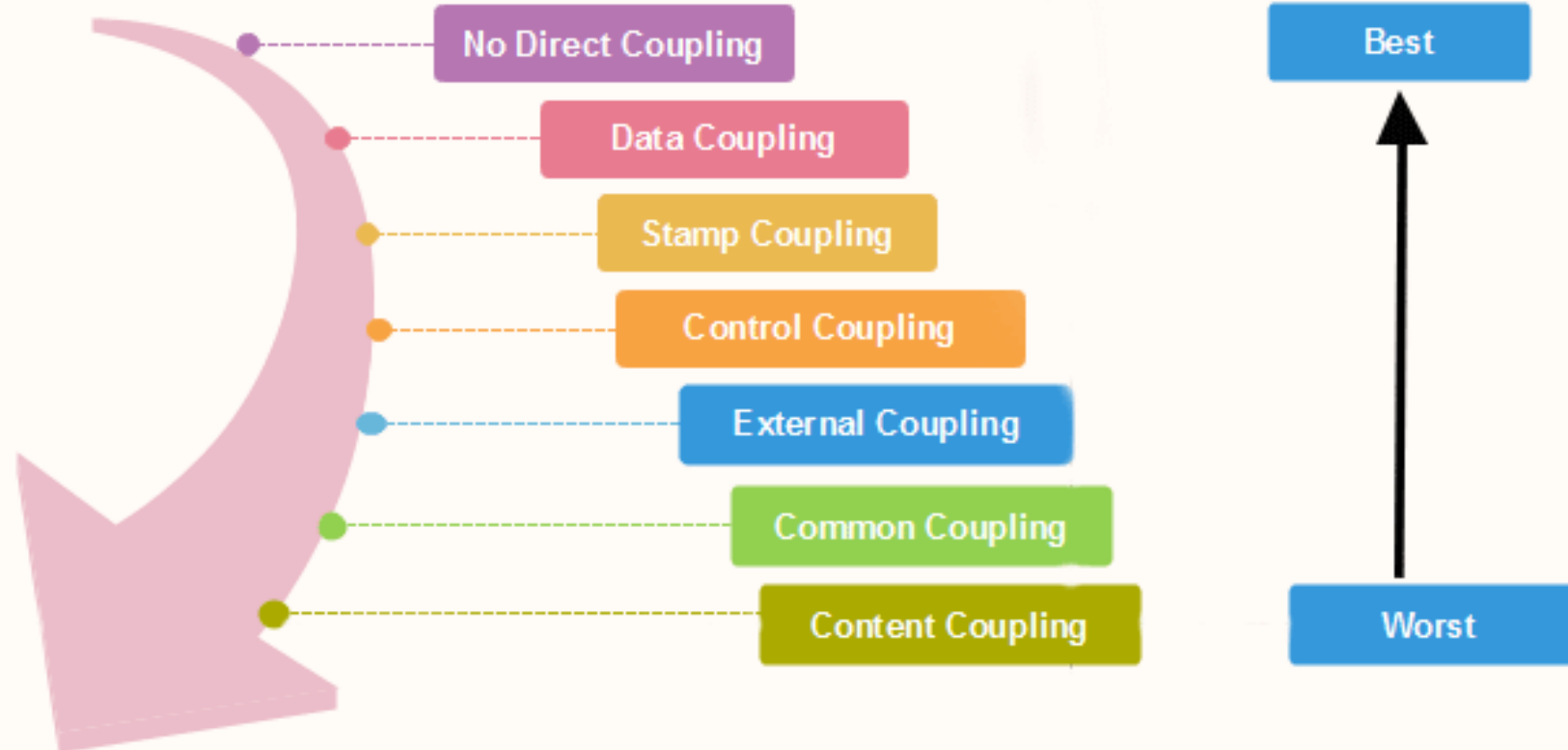
(c)

Coupling & Cohesion

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

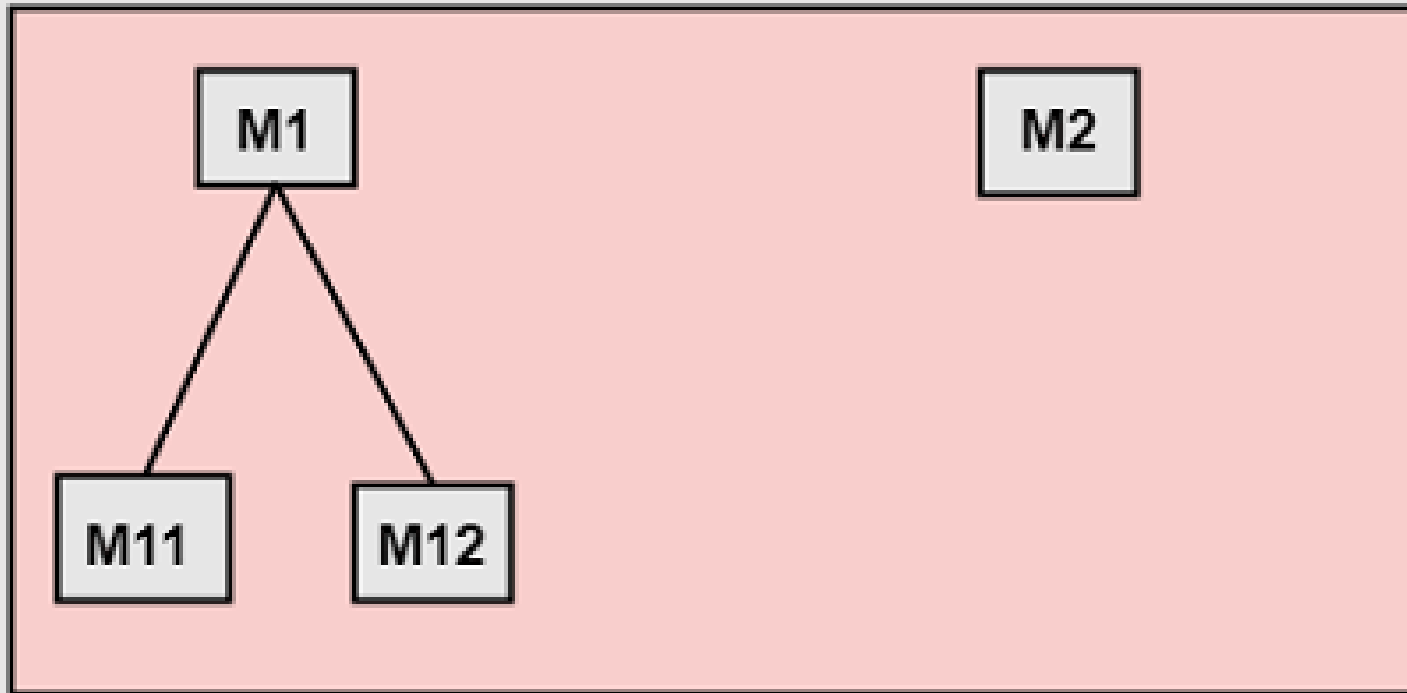
Types of Modules Coupling

There are various types of module Coupling are as follows:



Coupling & Cohesion

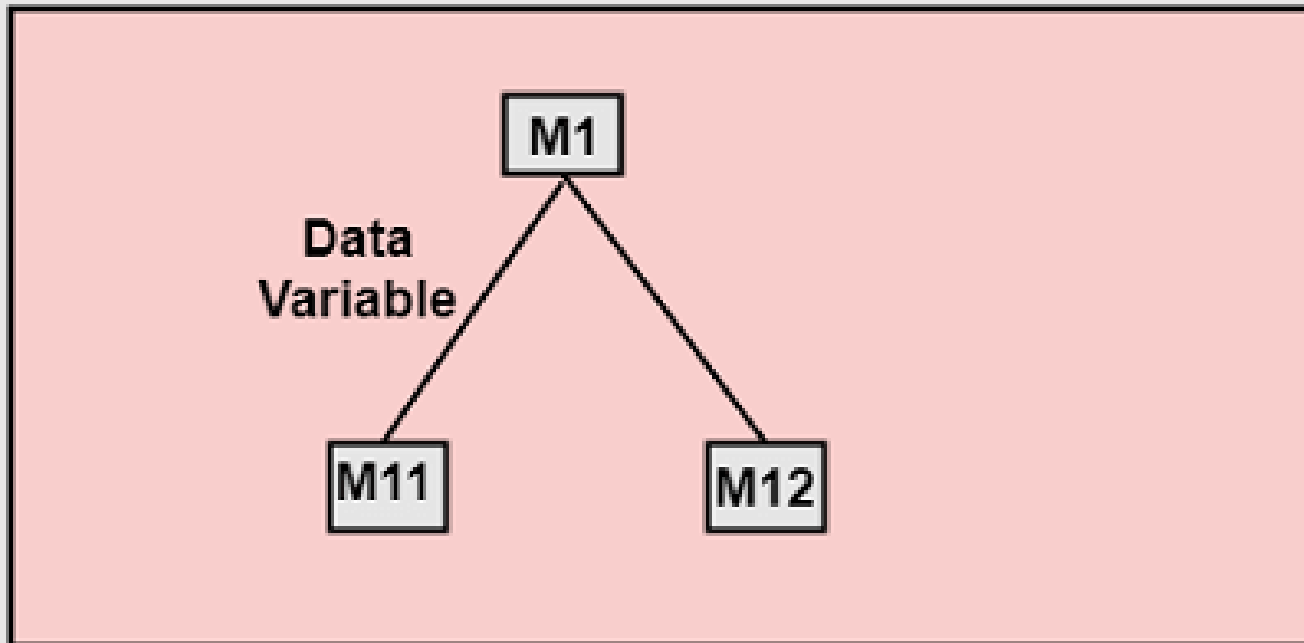
1. No Direct Coupling: There is no direct coupling between M1 and M2.



In this case, modules are subordinates to different modules.
Therefore, no direct coupling.

Coupling & Cohesion

2. Data Coupling: When data of one module is passed to another module, this is called data coupling.



Coupling & Cohesion

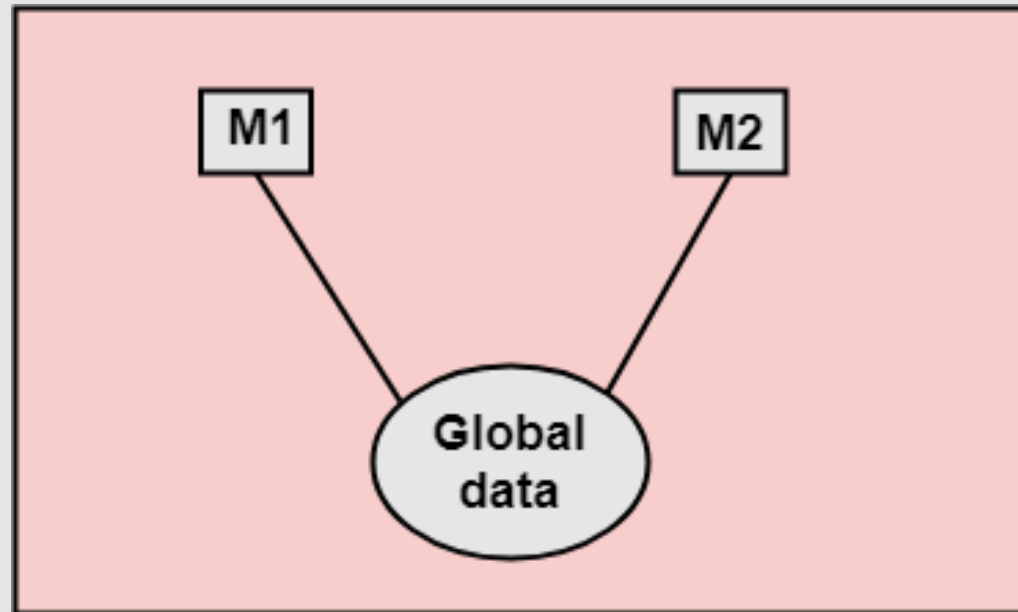
3. Stamp Coupling: Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object in C++ language to a module.

4. Control Coupling: Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.

5. 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.

Coupling & Cohesion

6. Common Coupling: Two modules are common coupled if they share information through some global data items.



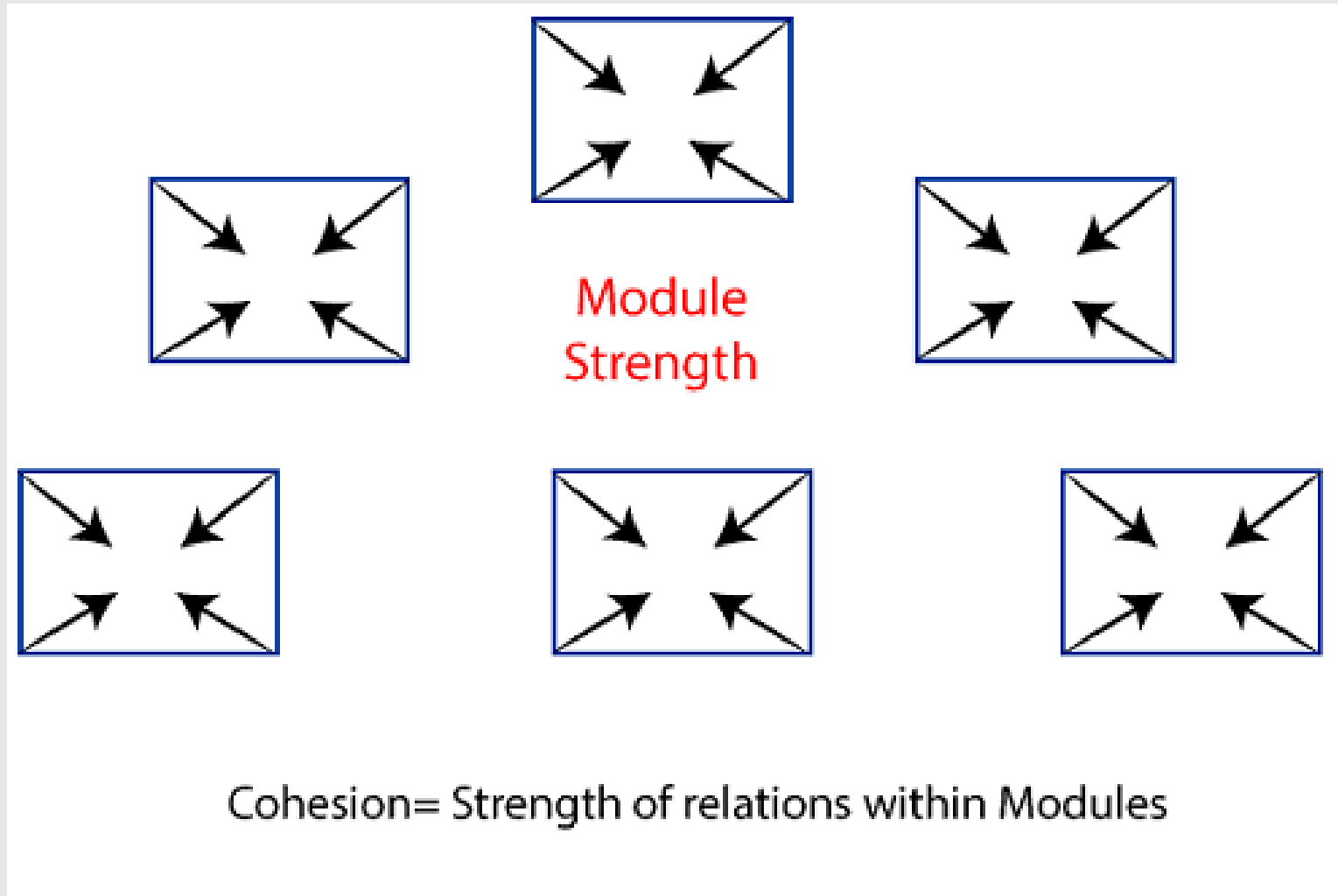
7. Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

Module Cohesion

In computer programming, cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related.

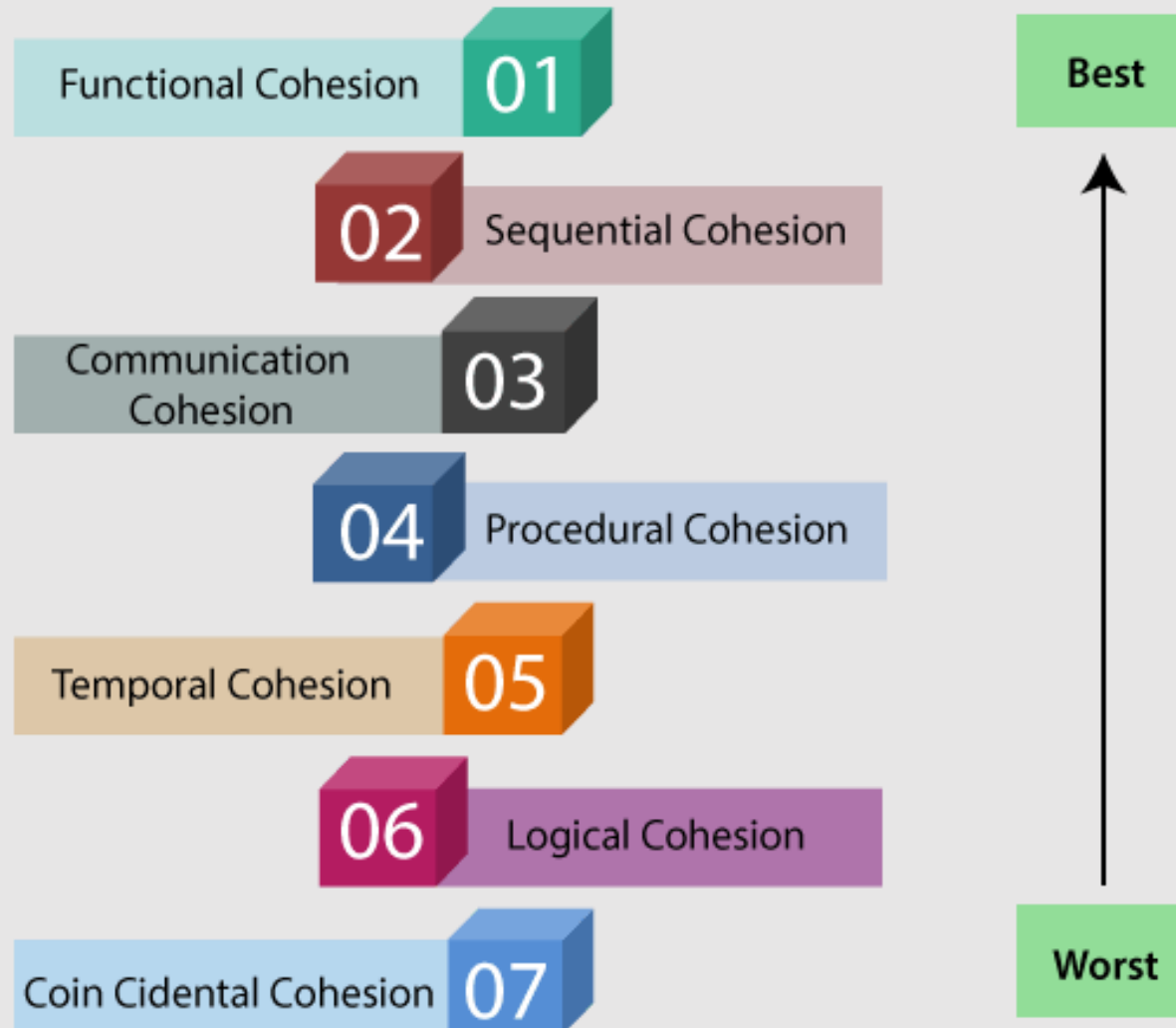
Cohesion is an ordinal type of measurement and is generally described as "high cohesion" or "low cohesion."

Module Cohesion



Module Cohesion

Types of Modules Cohesion



Types of Module Cohesion

Functional Cohesion: Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.

Sequential Cohesion: A module is said to possess sequential cohesion 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 Cohesion: A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.

Procedural Cohesion: A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.

Types of Module Cohesion

Temporal Cohesion: When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.

Logical Cohesion: A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.

Coincidental Cohesion: A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.

Difference between coupling and Cohesion

Differentiate between Coupling and Cohesion

Coupling	Cohesion
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.
Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.
Coupling shows the relative independence between the modules.	Cohesion shows the module's relative functional strength.
While creating, you should aim for low coupling, i.e., dependency among modules should be less.	While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.
In coupling, modules are linked to the other modules.	In cohesion, the module focuses on a single thing.

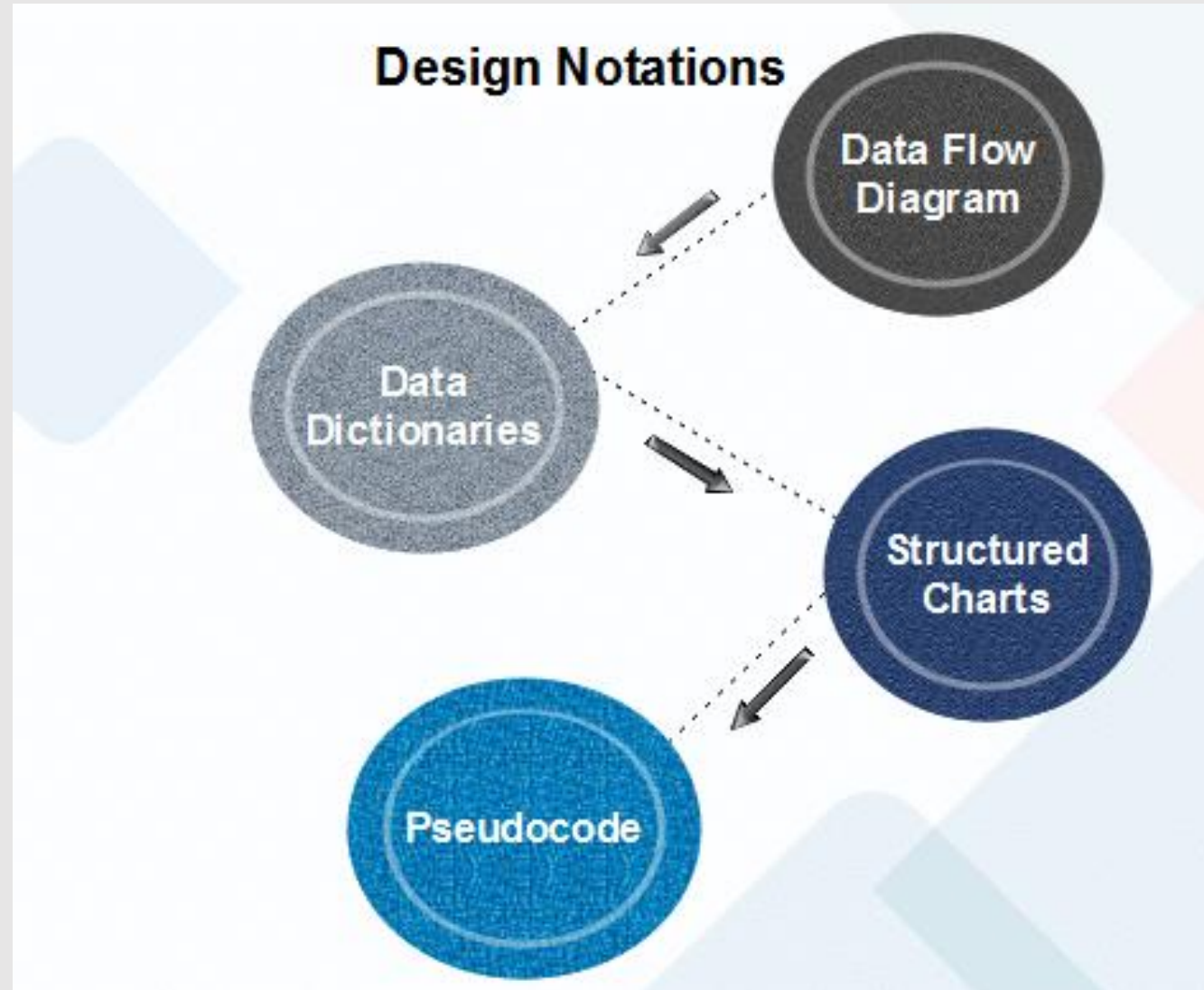
Function Oriented Design

Function Oriented design is a method to software design where the model is decomposed into a set of interacting units or modules where each unit or module has a clearly defined function. Thus, the system is designed from a functional viewpoint.

Design Notations

Design Notations are primarily meant to be used during the process of design and are used to represent design or design decisions. For a function-oriented design, the design can be represented graphically or mathematically by the following:

Function Oriented Design





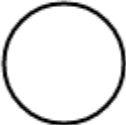

Data Flow Diagram

Data-flow design is concerned with designing a series of functional transformations that convert system inputs into the required outputs. The design is described as data-flow diagrams. These diagrams show how data flows through a system and how the output is derived from the input through a series of functional transformations.

Data-flow diagrams are a useful and intuitive way of describing a system. They are generally understandable without specialized training, notably if control information is excluded. They show end-to-end processing. That is the flow of processing from when data enters the system to where it leaves the system can be traced.

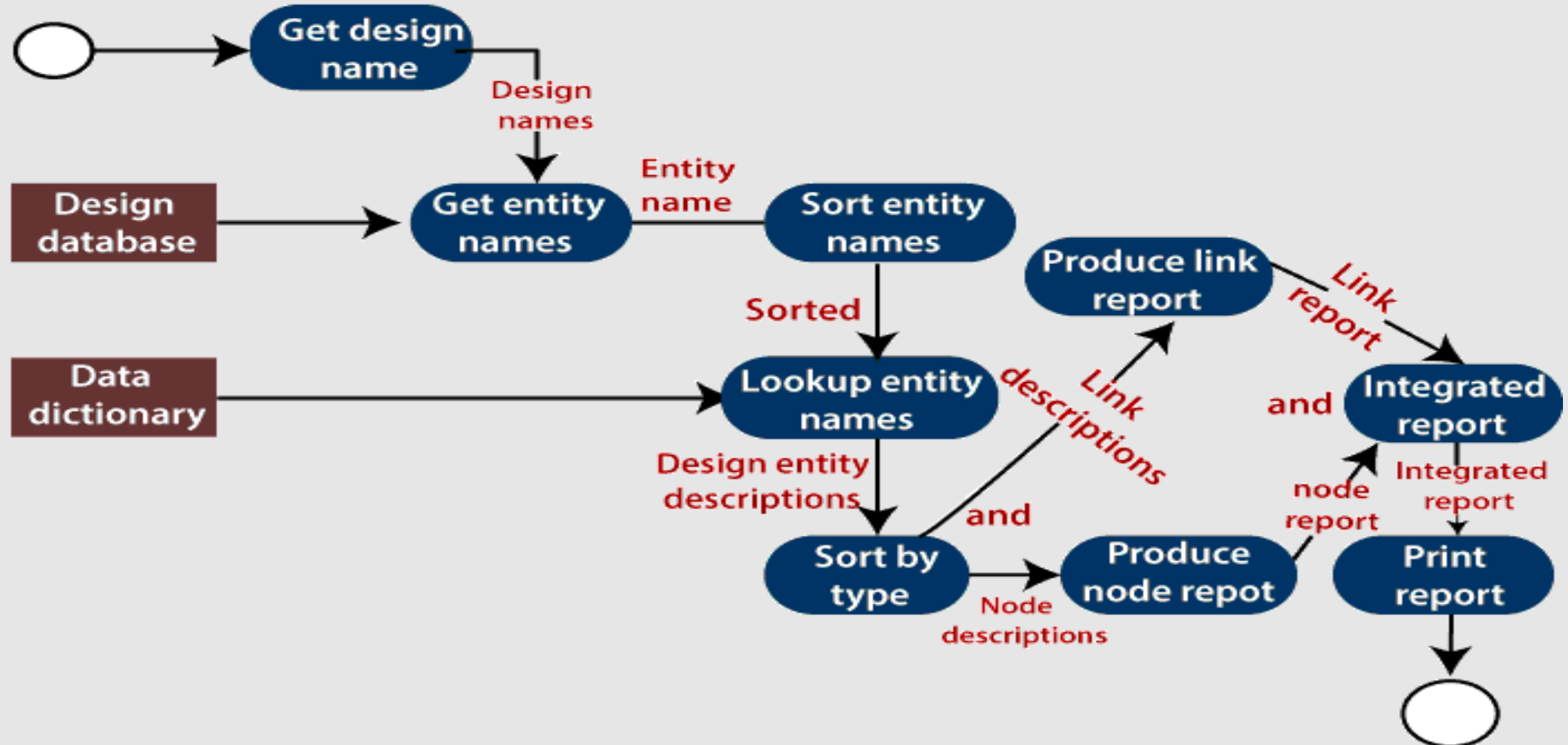
Data-flow design is an integral part of several design methods, and most CASE tools support data-flow diagram creation. Different ways may use different icons to represent data-flow diagram entities, but their meanings are similar.

Data Flow Diagram

Symbol	Name	Meaning
	Rounded Rectangle	It represents functions which transforms input to output. The transformation name indicates its function.
	Rectangle	It represents data stores. Again, they should give a descriptive name.
	Circle	It represents user interactions with the system that provides input or receives output.
	Arrows	It shows the direction of data flow. Their name describes the data flowing along the path.
"and" and "or"	Keywords	The keywords "and" and "or". These have their usual meanings in boolean expressions. They are used to link data flows when more than one data flow may be input or output from a transformation.

Data Flow Diagram of a design report generator

Data flow diagram of a design report generator



Report Generator

The report generator produces a report which describes all of the named entities in a data-flow diagram. The user inputs the name of the design represented by the diagram. The report generator then finds all the names used in the data-flow diagram. It looks up a data dictionary and retrieves information about each name. This is then collated into a report which is output by the system.

Report Generator

The report generator produces a report which describes all of the named entities in a data-flow diagram. The user inputs the name of the design represented by the diagram. The report generator then finds all the names used in the data-flow diagram. It looks up a data dictionary and retrieves information about each name. This is then collated into a report which is output by the system.

Data Dictionaries

A data dictionary plays a significant role in any software development process because of the following reasons:

- A Data dictionary provides a standard language for all relevant information for use by engineers working in a project. A consistent vocabulary for data items is essential since, in large projects, different engineers of the project tend to use different terms to refer to the same data, which unnecessarily causes confusion.
- The data dictionary provides the analyst with a means to determine the definition of various data structures in terms of their component elements.

Structured Charts

It partitions a system into block boxes. A Black box system that functionality is known to the user without the knowledge of internal design.

Structured Chart is a graphical representation which shows:

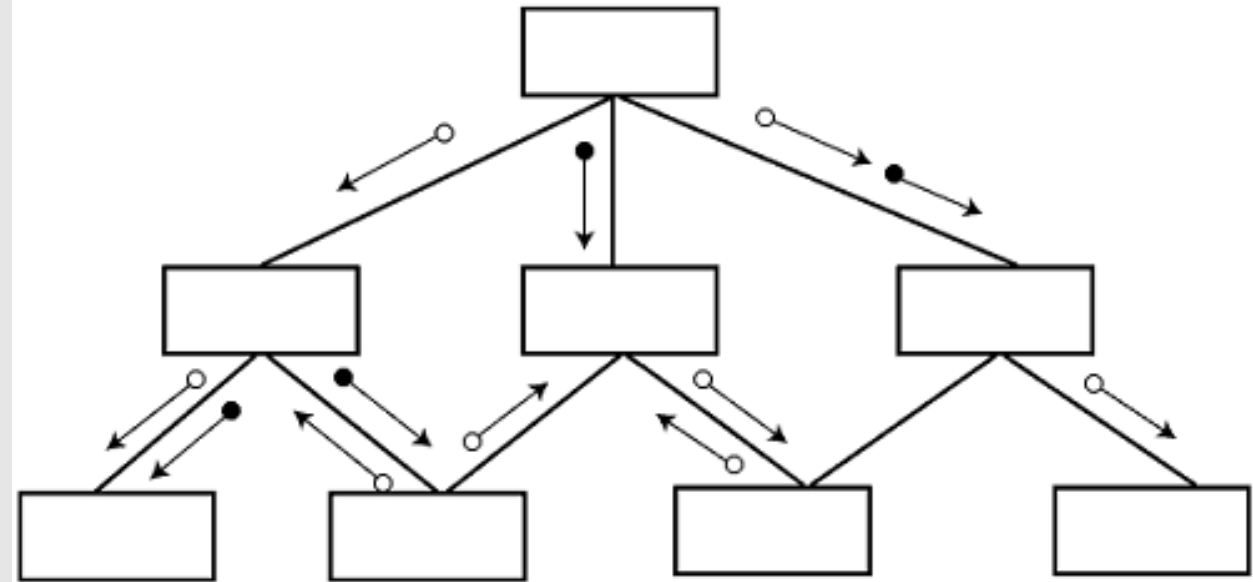
System partitions into modules

Hierarchy of component modules

The relation between processing modules

Interaction between modules





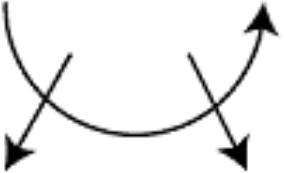
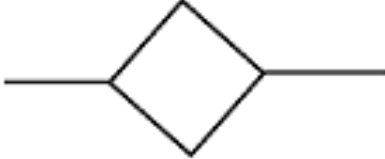
Information passed between modules



Hierarchical format of a structure chart

Structured Charts

The following notations are used in structured chart:

SYMBOL	DESCRIPTION
	Module
	Arrow
	Data couple
	Control Flag
	Loop
	Decision

Pseudo-code

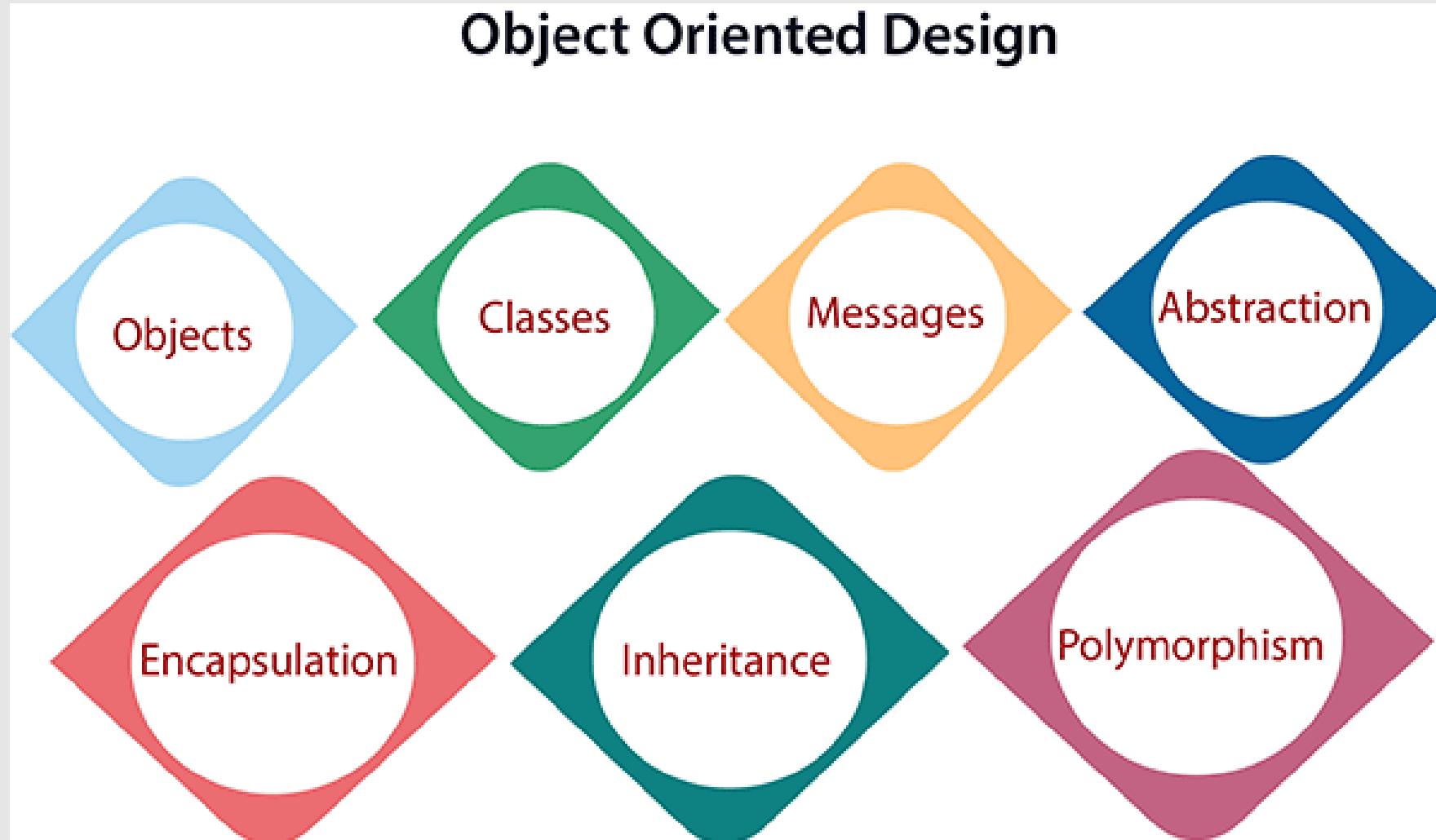
Pseudo-code notations can be used in both the preliminary and detailed design phases. Using pseudo-code, the designer describes system characteristics using short, concise, English Language phrases that are structured by keywords such as If-Then-Else, While-Do, and End.

Object – oriented Design

In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data. For example, in a Library Automation Software, each library representative may be a separate object with its data and functions to operate on these data. The tasks defined for one purpose cannot refer or change data of other objects. Objects have their internal data which represent their state. Similar objects create a class. In other words, each object is a member of some class. Classes may inherit features from the superclass.

Object – oriented Design

The different terms related to object design are:



Object – oriented Design

Objects: All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.

Classes: A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.

Messages: Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.

Abstraction In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.ent terms related to object design are:

Object – oriented Design

Encapsulation: Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.

Inheritance: OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate superclasses.

This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.

Polymorphism: OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.

User Interface Design

The visual part of a computer application or operating system through which a client interacts with a computer or software. It determines how commands are given to the computer or the program and how data is displayed on the screen.

Types of User Interface

There are two main types of User Interface:

- Text-Based User Interface or Command Line Interface
- Graphical User Interface (GUI)

Text-Based User Interface: This method relies primarily on the keyboard. A typical example of this is UNIX.

User Interface Design

Advantages

Many and easier to customizations options.
Typically capable of more important tasks.

Disadvantages

Relies heavily on recall rather than recognition.
Navigation is often more difficult.

User Interface Design

Graphical User Interface (GUI): GUI relies much more heavily on the mouse. A typical example of this type of interface is any versions of the Windows operating systems.

GUI Characteristics

Characteristics	Descriptions
Windows	Multiple windows allow different information to be displayed simultaneously on the user's screen.
Icons	Icons different types of information. On some systems, icons represent files. On other icons describes processes.
Menus	Commands are selected from a menu rather than typed in a command language.
Pointing	A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interests in a window.
Graphics	Graphics elements can be mixed with text or the same display.

User Interface Design

Advantages

Less expert knowledge is required to use it.

Easier to Navigate and can look through folders quickly in a guess and check manner.

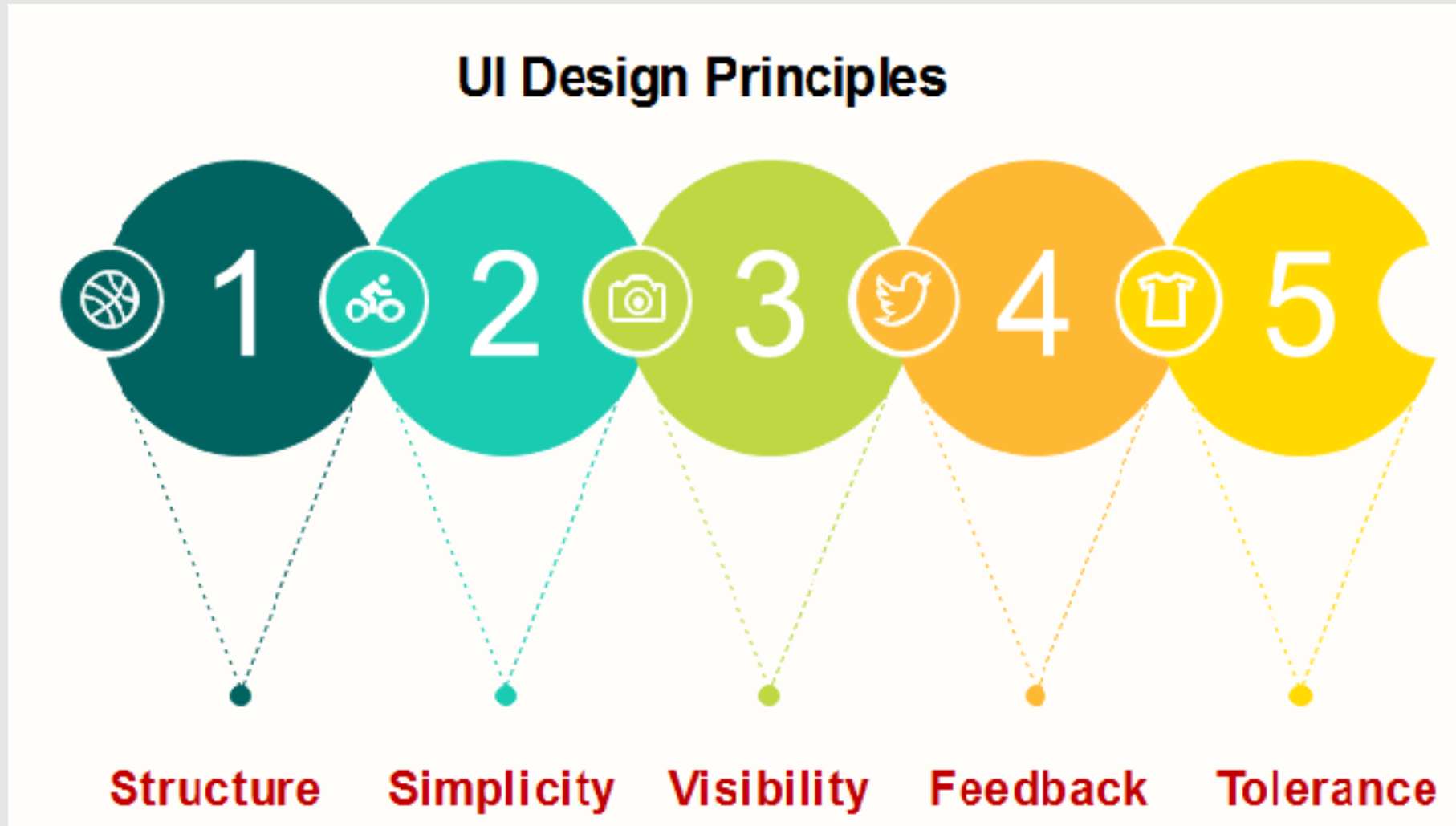
The user may switch quickly from one task to another and can interact with several different applications.

Disadvantages

Typically decreased options.

Usually less customizable. Not easy to use one button for tons of different variations.

UI Design Principles



UI Design Principles

Structure: Design should organize the user interface purposefully, in the meaningful and usual based on precise, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.

Simplicity: The design should make the simple, common task easy, communicating clearly and directly in the user's language, and providing good shortcuts that are meaningfully related to longer procedures.

Visibility: The design should make all required options and materials for a given function visible without distracting the user with extraneous or redundant data.

UI Design Principles

Feedback: The design should keep users informed of actions or interpretation, changes of state or condition, and bugs or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.

Tolerance: The design should be flexible and tolerant, decreasing the cost of errors and misuse by allowing undoing and redoing while also preventing bugs wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.

THANK YOU!!!