SOFTWARE ENGINEERING
(Week-6)

USAMA MUSHARAF

MS-CS (Software Engineering)

LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

AGENDA OF WEEK # 6

Types of Coupling & Cohesion (Cont..)

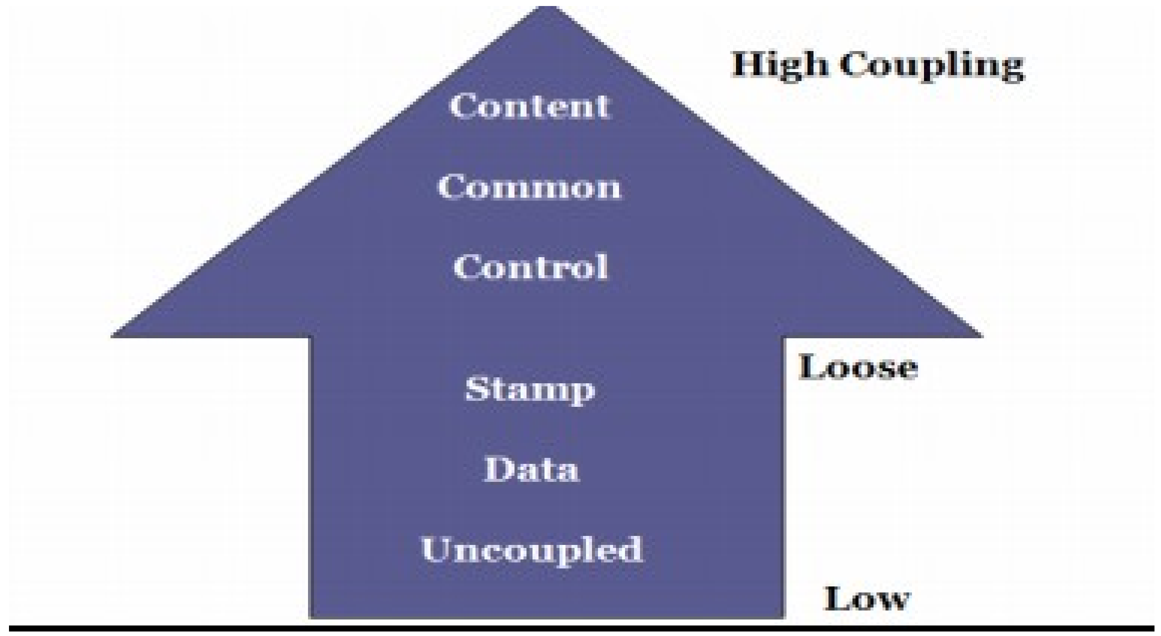
Conceptual Model of Architecture Representation

Architectural Views

Views and View Point

4+1 View Model

TYPES OF COUPLING

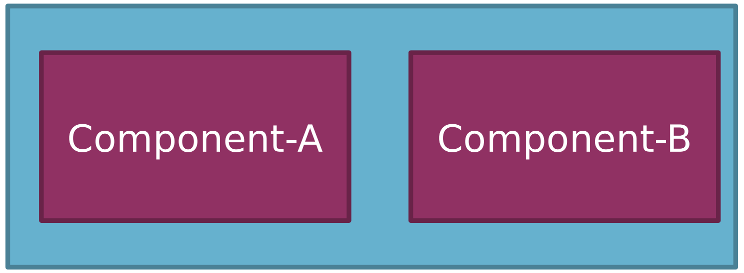


CONTENT COUPLING

This is highest level of coupling and occurs if there are two (or more) modules and if one refers to the ``inside'' - the ``internal'' or ``private'' part - of the other in some way.

Component A handles lookup data.

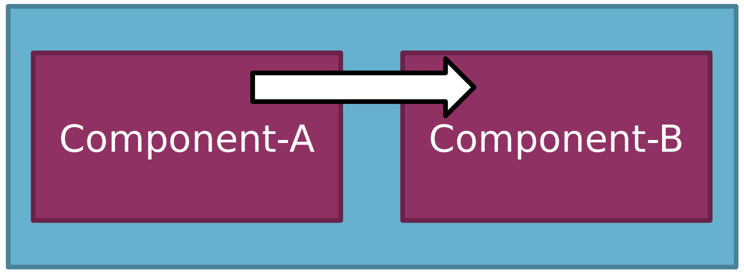
Component is handling the property of adding new customer when needed.



CONTENT COUPLING EXAMPLE

When we are searching the customer data and customer is not found, lookup component adds customer by directly modifying the contents of the data structure containing customer data (new customer).

This is a very high level of coupling in which one component is directly modifying the content of another component and this is not wanted in any software design.



CONTENT COUPLING EXAMPLE

Solution:

When customer not found, component calls the AddCustomer () method that is responsible for maintaining customer data rather than directly modifying data structure.

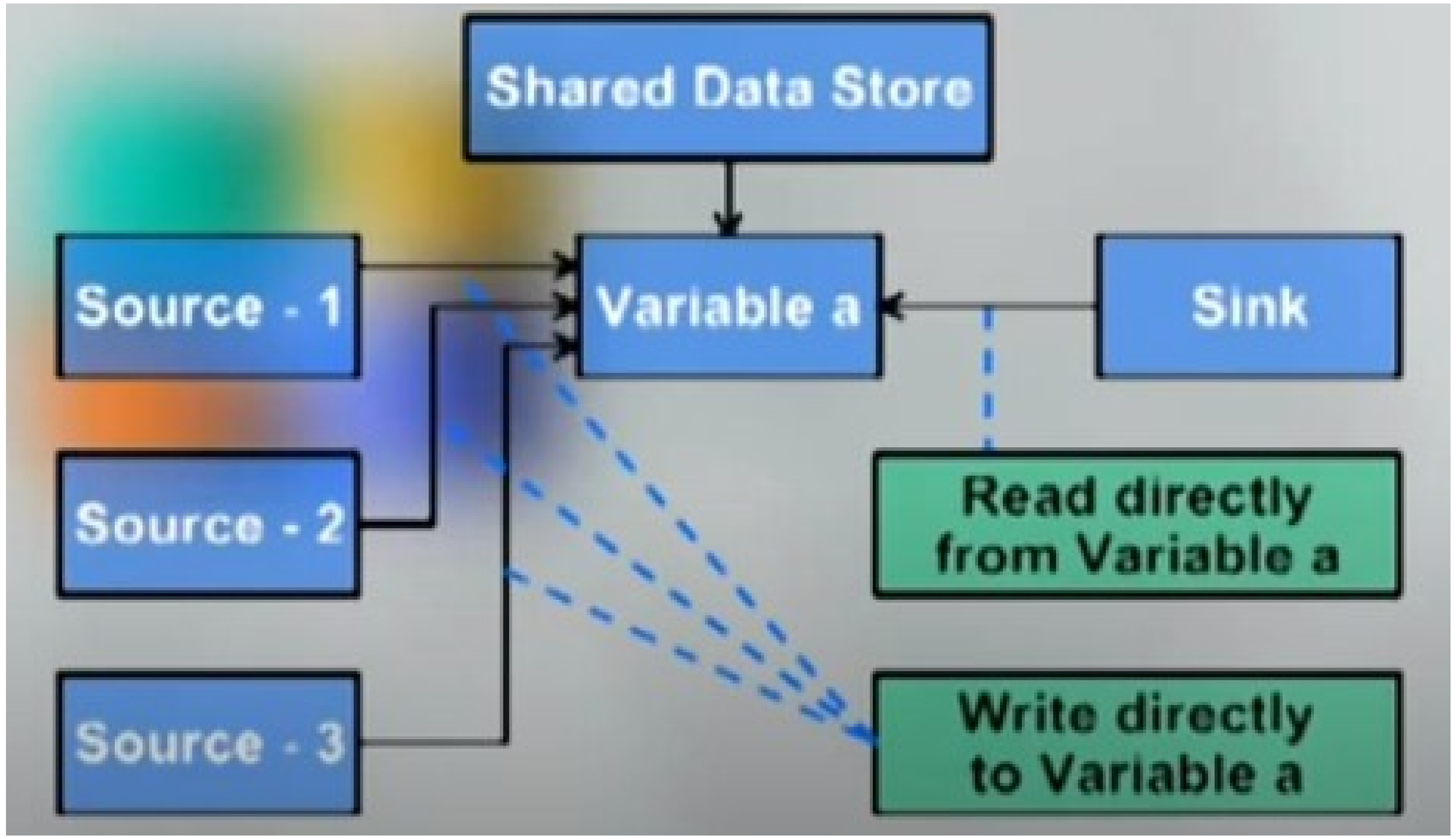
COMMON COUPLING

Common coupling occurs when modules communicate using global data structures.

For example, programming allows the developer to declare a data element as external, enabling it to be accessed by all modules.

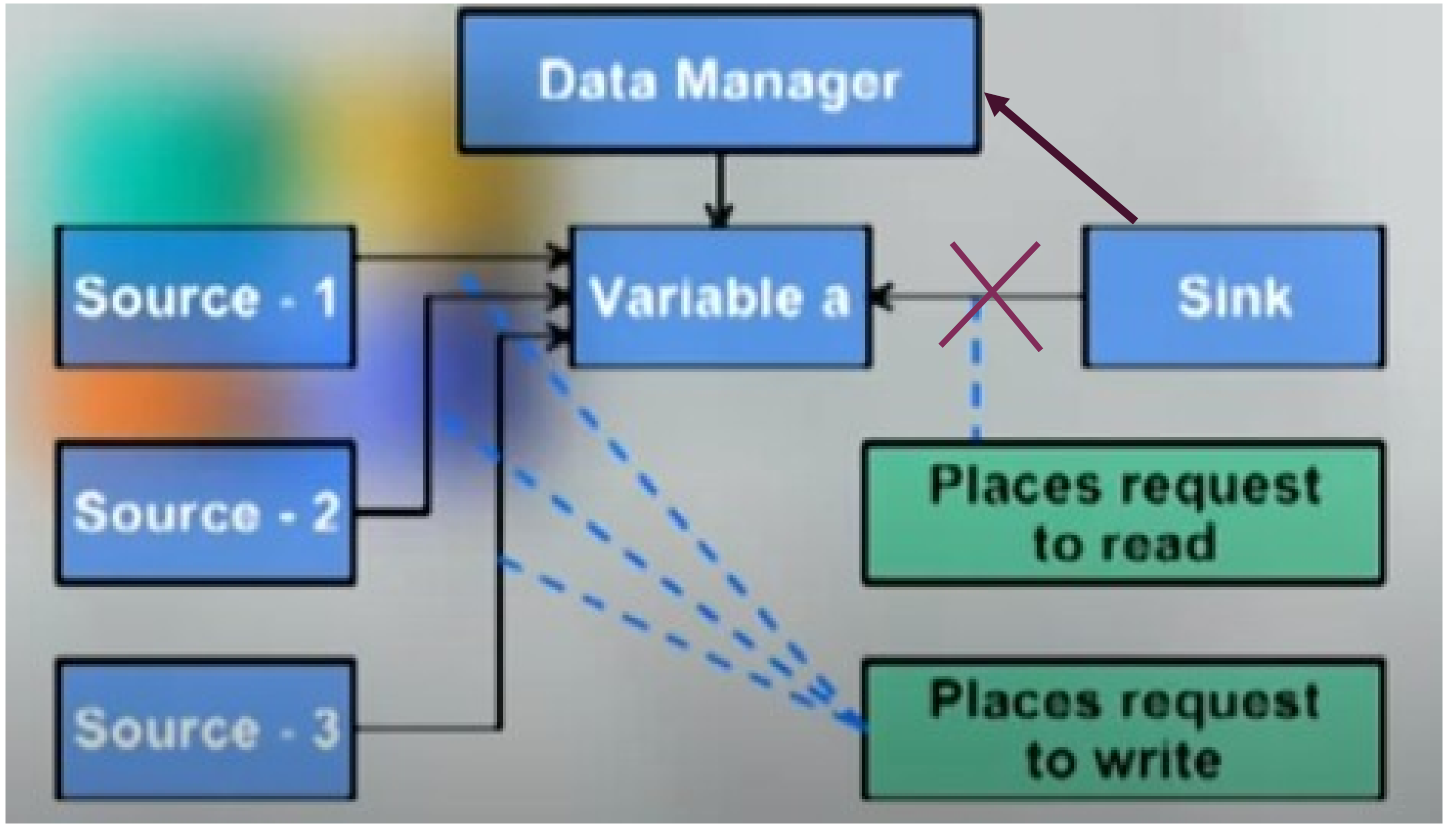
Common coupling is also known as “Global coupling”. We can say that two components share data using Global data structures.

COMMON COUPLING EXAMPLE



COMMON COUPLING

Solution:



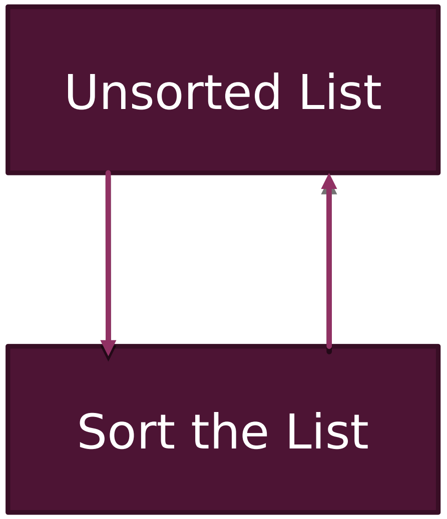
CONTROL COUPLING (MODERATE COUPLING)

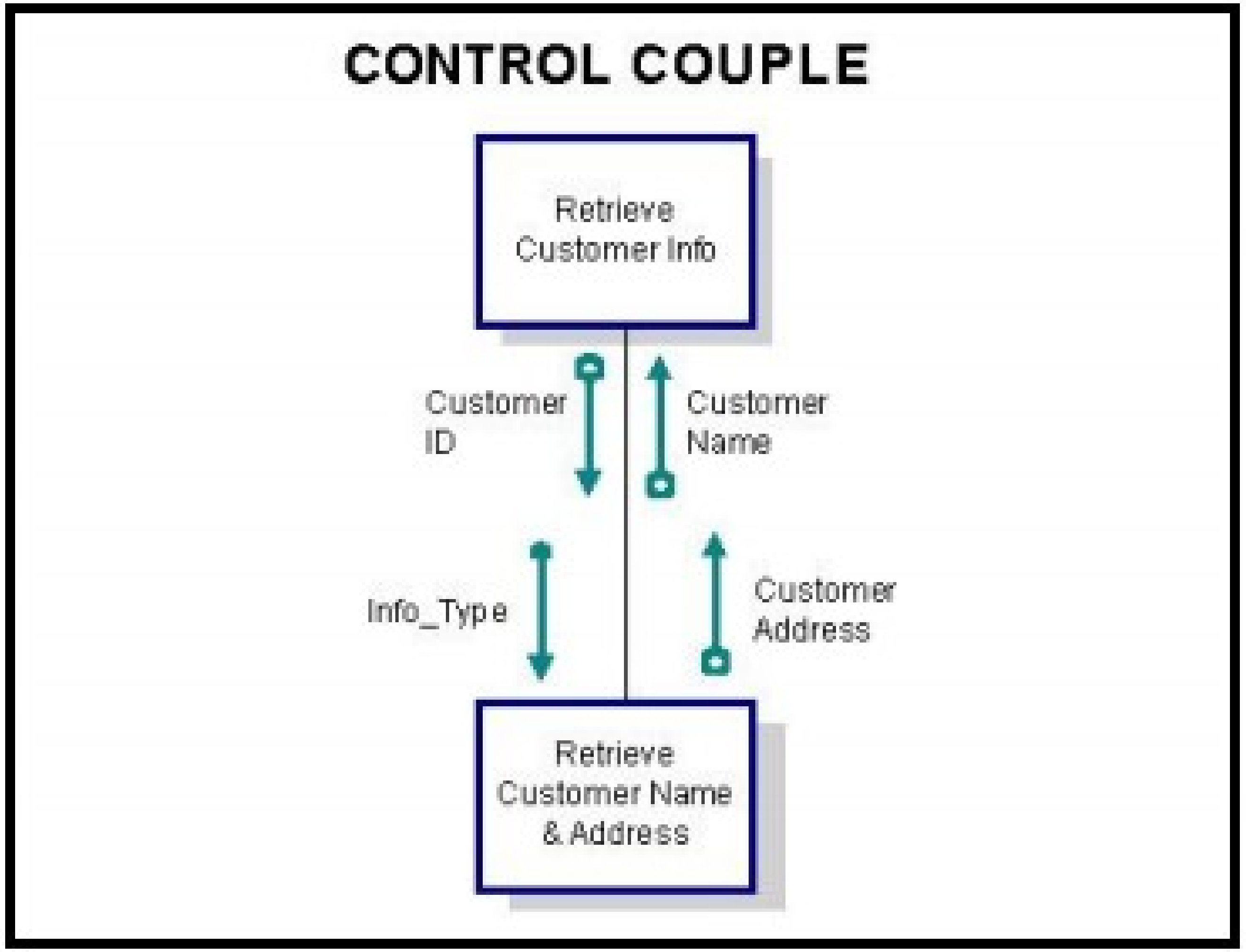
Two modules exhibit control coupling if one (``module A'') passes to the other (``module B'') a piece of information that is intended to control the internal logic of the other.

In control coupling, we need to take care off components when it comes to testing.

Our Focus should be on integration testing.

CONTROL COUPLING EXAMPLE





STAMP COUPLING

Beginning of Low level coupling.

Stamp coupling is when modules share a composite data structure and use only a part of it, possibly a different part (e.g., passing a whole record to a function that only needs one field of it)

or

Two modules (``A'' and ``B'') exhibit stamp coupling if one passes directly to the other a ``composite'' piece of data - that is, a piece of data with meaningful internal structure - such as a record (or structure), array, or (pointer to) a list or tree.

STAMP COUPLING EXAMPLE

Example: Customer Billing System

The print routine of the customer billing accepts a customer data structure (cid, customer name, address, phone no, email address, NIC, cell no, passport no) as an argument, parses it, and prints the name, address, and billing amount.

The billing amount is retrieved on the basis of cid which is passed as an argument in the module.

Only cid, customer name and address is used other attributes are not used or in other word they are not needed.

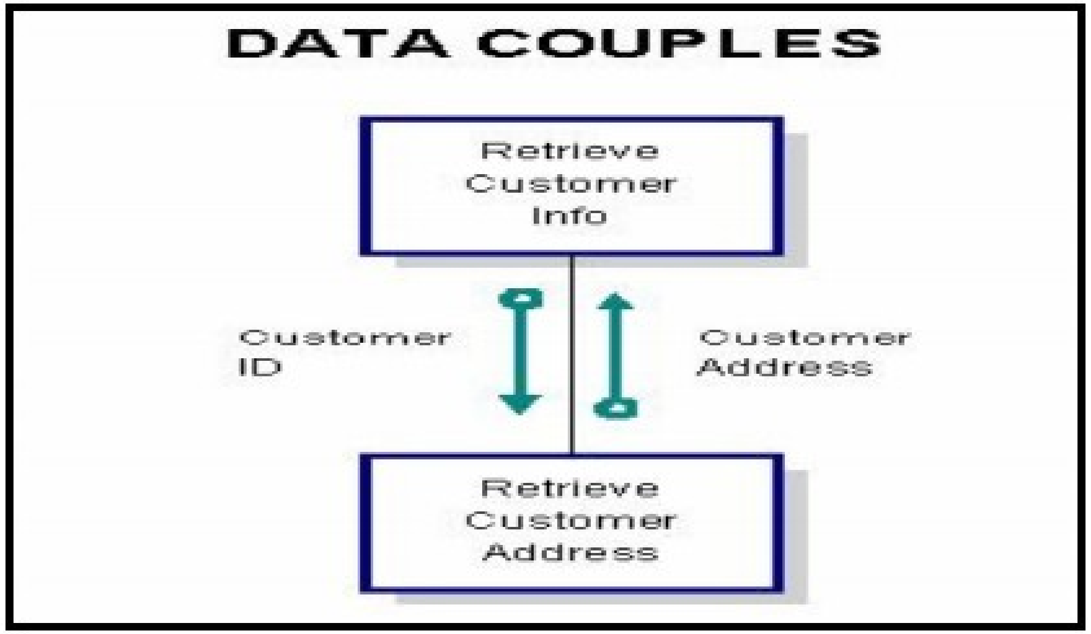
STAMP COUPLING SOLUTION

The print routine takes the customer name, address, and billing information as an argument rather than taking the whole structure as input.

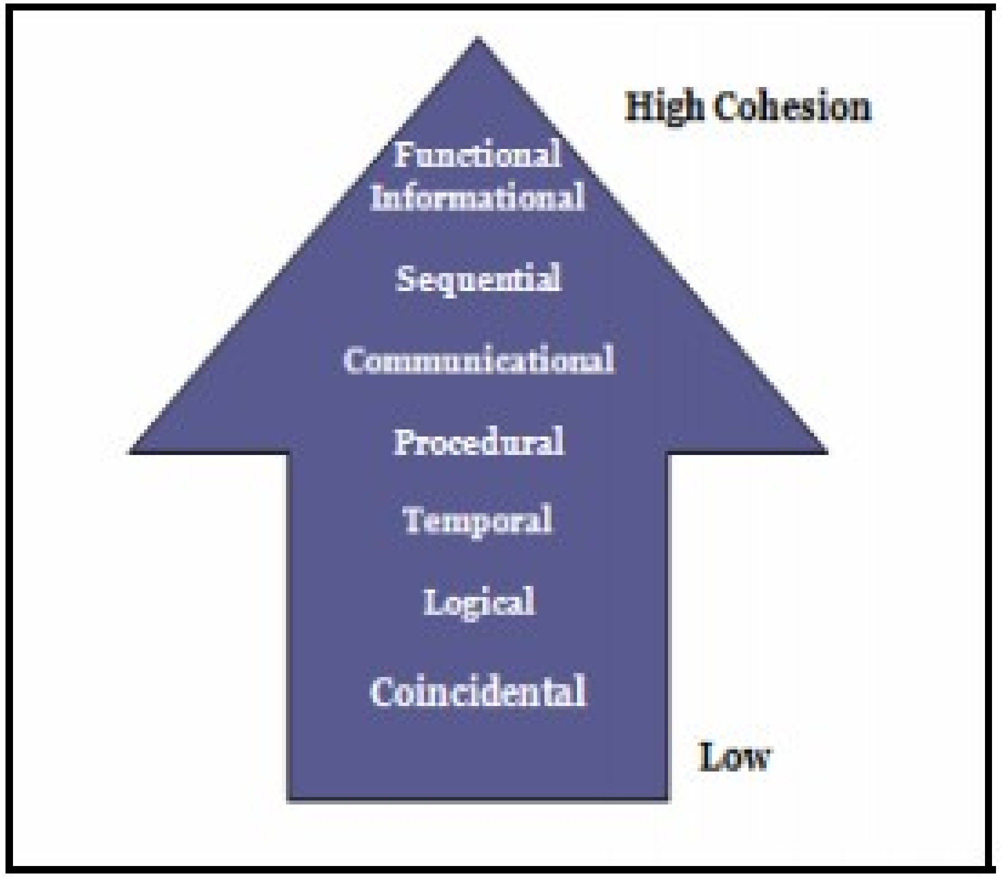
DATA COUPLING – LOW LEVEL OF COUPLING

Data coupling occurs between two modules when data are passed by parameters using a simple argument list and every item in the list is used.

Example:



TYPES OF COHESION



COINCIDENTAL COHESION- LOWEST COHESION

Coincidental cohesion occurs when parts of the component are only related by their location in source code.

A coincidentally cohesive module is one whose activities have no meaningful relationship to one another.

COINCIDENTAL COHESION- LOWEST COHESION

Coincidental cohesion is considered the worst level of cohesion.

Its activities are not related by flow of data or by flow of control.

COINCIDENTAL COHESION- LOWEST COHESION

Example:

Print next line

Reverse string of characters in second argument

Add 7 to 5th argument.

Convert 4th argument to float.

All the above functions have no relationship but they are there in the same module by chance!!

LOGICAL COHESION

In this level of cohesion elements of component are related logically and not functionally.

Example:

In performing an I/O operation a component reads inputs from tape, disk, and network.

All the code for these functions is in the same component.

It is to note that operations are related, but the functions are significantly different i-e the way to execute each function is different by the different ways of performing I/O belong to same category that’s why they are in same module.

TEMPORAL COHESION

A temporally cohesive module is one which performs several activities that are related in time.

Temporally cohesive modules typically consist of partial activities whose only relationship to one another is that they are all carried out at a specific time.

TEMPORAL COHESION

Example:

Consider a module called "On_Really_Bad_Failure" that is invoked when a Really_Bad_Failure happens.

The module performs several tasks that are not functionally similar or logically related, but all tasks need to happen at the moment when the failure occurs.

The module might

i. Cancel all outstanding requests for services
ii. Notify the operator console of the failure
iii. Make an entry in a database of failure records

PROCEDURAL COHESION

Elements of a component are related only to ensure a particular order of execution.

In this type of cohesion the elements are arranged together in the module but with a condition that there is particular sequence to be followed to execute them.

Activities in a procedurally cohesive module are related by flow of execution rather than by one problem-related function.

PROCEDURAL COHESION

Example:

Suppose there is a module which intents to repair the damaged record of the database and then update the maintenance file.

In this kind of application we can’t repair the damaged record without reading it from the database; we can only repair the record after reading it from the database. So the order the execution may be:

Read data from data base.

Update repair record.

Create an entry in log file of maintenance.

COMMUNICATIONAL COHESION

In communicational cohesion, module performs a series of actions related by a sequence of steps to be followed by the product and all actions are performed on the same data.

A communicational cohesive module is one which performs several functions on the same input or output data.

COMMUNICATIONAL COHESION

Example:

- Read record from database
- Process data (Gpa calculation)
- Generate Report (Print)

SEQUENTIAL COHESION

A sequentially cohesive module contains activities where output data from one activity serves as input data to the next activity.

In general, a sequentially cohesive module has good coupling and is easy to maintain.

SEQUENTIAL COHESION

Example

Suppose we want to edit the customer data we need to perform the following tasks in sequence:

Retrieve customer Data.

Retrieve customer order.

Generate invoice.

Get and edit input data.

FUNCTIONAL COHESION- HIGHEST COHESION

A functionally cohesive module performs one and only one problem related task and every essential element to a single computation is contained in the component.

A functionally cohesive module performs one and only one problem related task.

This is ideal situation.

FUNCTIONAL COHESION- HIGHEST COHESION

Functionally cohesive modules are good candidates for re-use and systems built with functionally cohesive modules are easily understood and, therefore, easier to maintain.

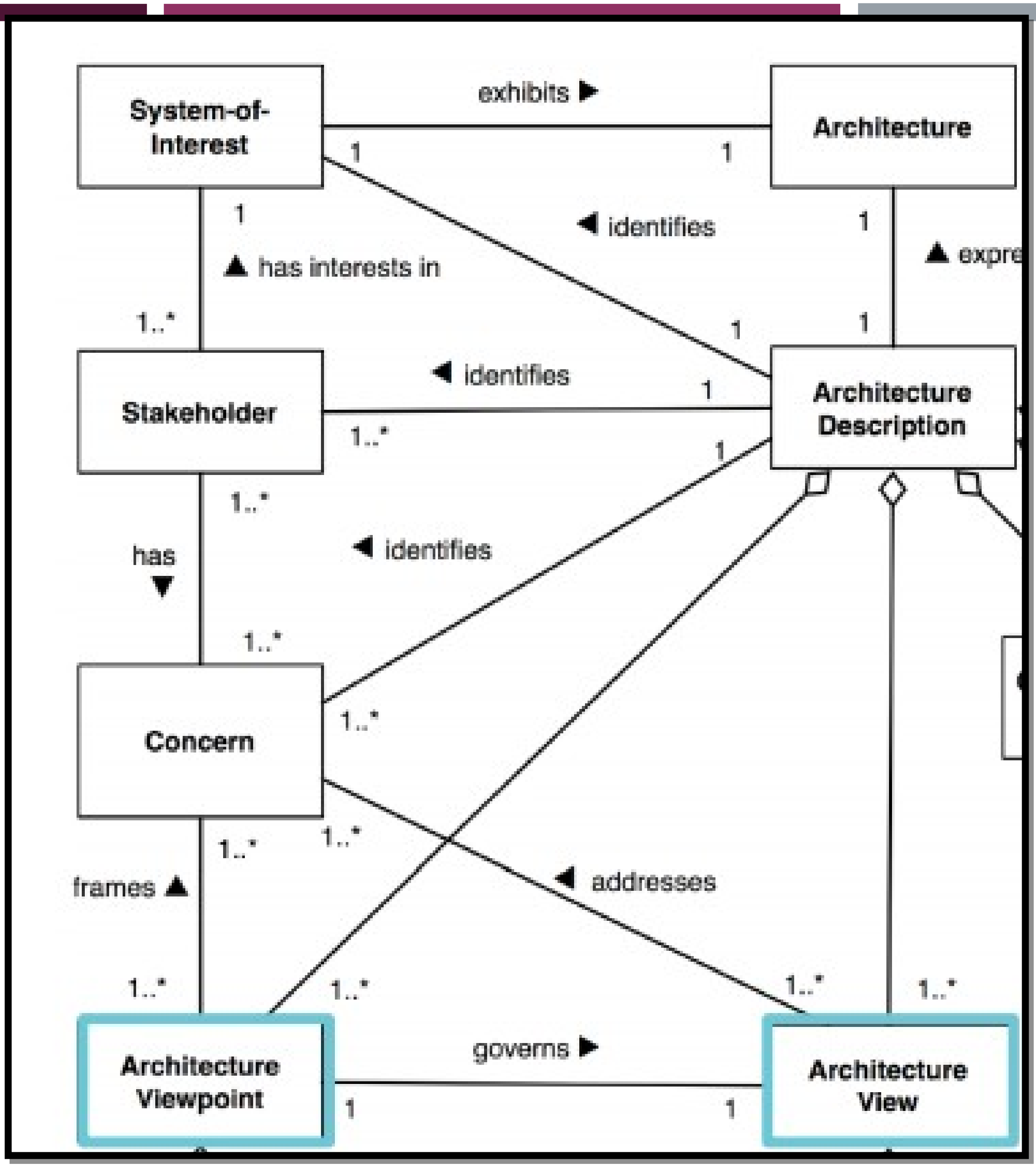
Example = ?

“

Conceptual Model of an Architecture Description

”

ISO/IEC/IEEE 42010


System-of-Interest
exhibits ▶
Architecture
1
1
1
◀ identifies
1
▲ has interests in
▲ expre
1..*
1
1
Stakeholder
◀ identifies
1
Architecture Description
1..*
1
1..*
has ▼
◀ identifies
1..*
Concern
1..*
1..*
1..*
frames ▲
◀ addresses
1..*
1..*
1..*
1..*
Architecture Viewpoint
governs ▶
Architecture View
1
1

INTRODUCTION TO VIEWS

Dictionary Meaning

Manner of looking at something

Why (multiple) view ?

For better understanding and managing.

Multi dimensional view must be taken for any complex entity because of its complex nature ,

It can’t be described in 1 dimensional view.

INTRODUCTION TO VIEWS

For example, In civil what are the views of a building…

- *Room layout*
- *3D view of building / room*
- *Electrical diagram*
- *Plumbing diagram*
- *Security alarm diagram*
- *AC duct diagram etc…etc…*

- Which of the above view is Architecture?
- **In Software, What are views ? ……….**

DEFINITION OF SW VIEW

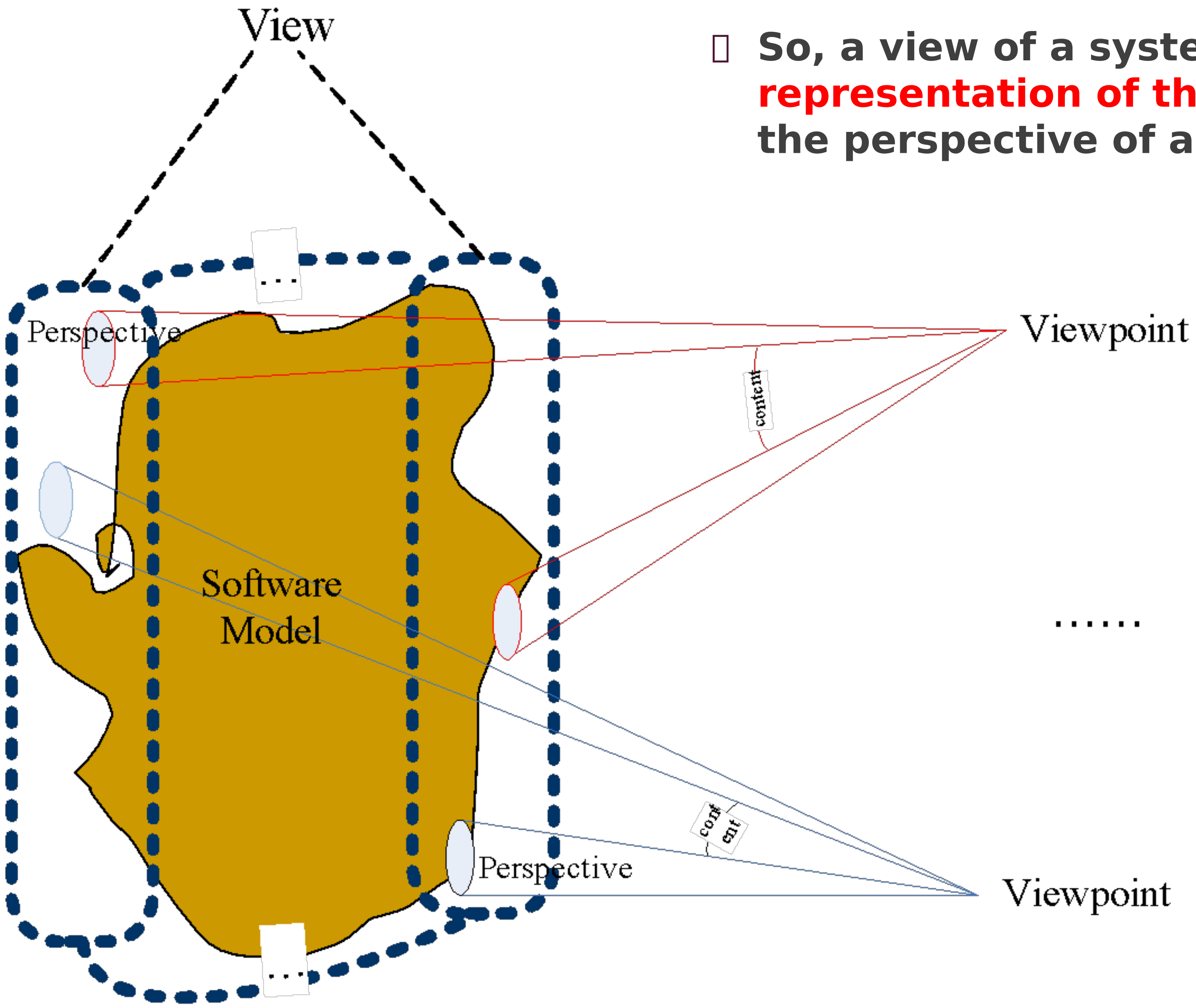
As per IEEE definition,

Software architecture descriptions are commonly organized into views,

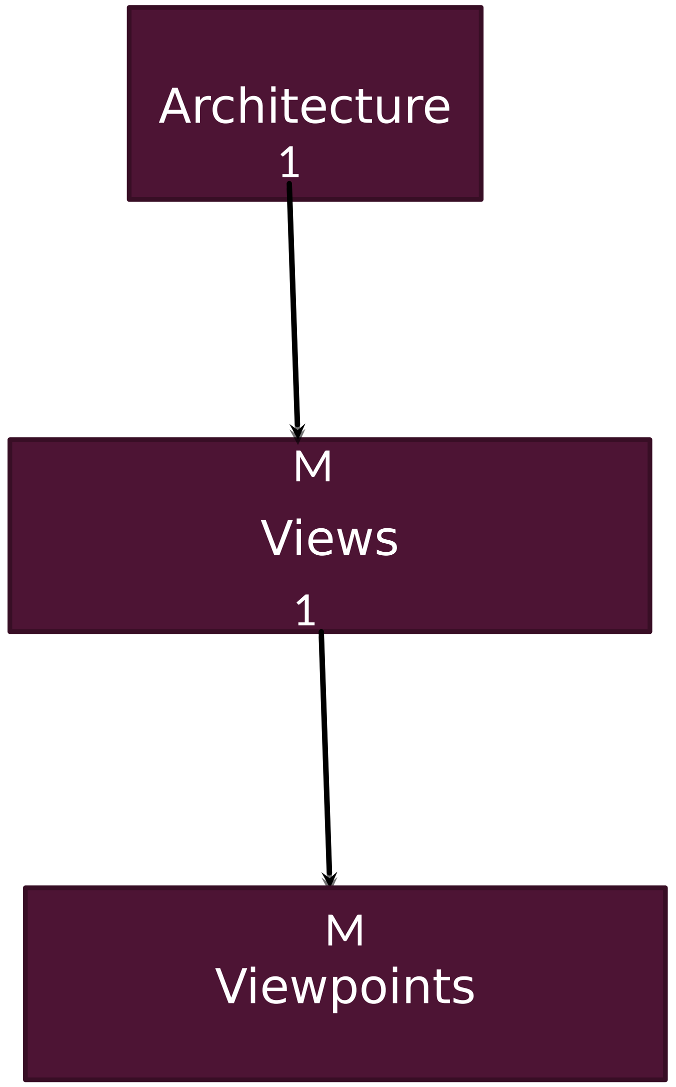
Each view addresses a set of system concerns, following the conventions of its *viewpoint*.

Viewpoint - A position or direction from which something is observed or considered;

View – Details or full specification considered from that viewpoint



- So, a view of a system is **a representation of the system** from the perspective of a viewpoint.


Architecture
1
M
Views
1
M
Viewpoints

VIEW MODEL

Software designers can organize the description of their architecture decisions in different views.

4+1 VIEW MODEL

The 4+1 view is an architecture verification technique for studying and documenting software architecture design.

THE 4 +1 VIEW MODEL

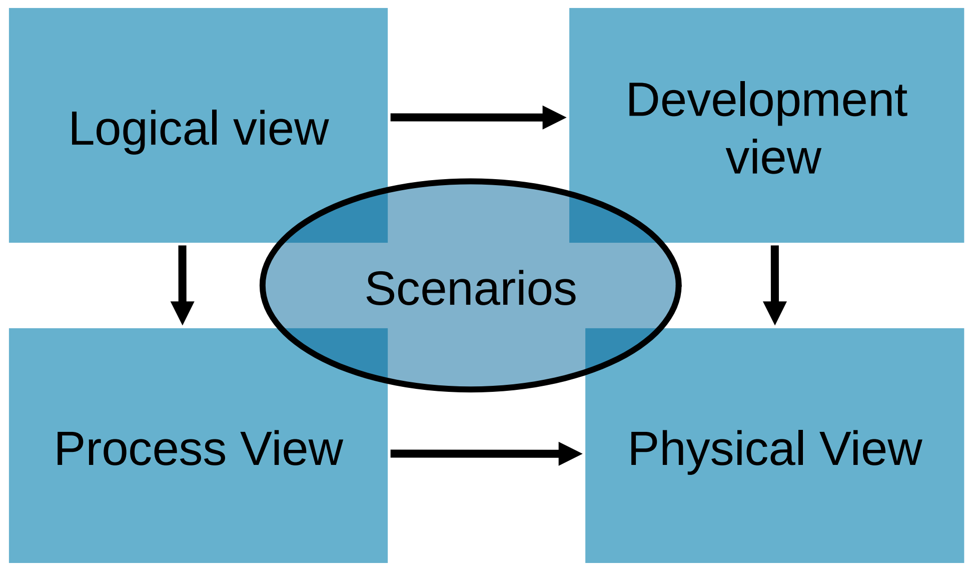
The 4+1 view model was originally introduced by Philippe Kruchten (Kruchten, 1995).

The model provides four essential views:

- the logical view,
- the process view,
- the physical view,
- the development view

and fifth is the scenario view

4+1 VIEW MODEL OF ARCHITECTURE



THE 4+1 VIEW MODEL

Multiple-view model that addresses different aspects and concerns of the system.

Standardizes the software design documents and makes the design easy to understand by all stakeholders.

THE SCENARIO VIEW- USE CASE VIEW

The scenario view describes the functionality of the system, i.e., how the user employs the system and how the system provides services to the users.

It helps designers to discover architecture elements during the design process and to validate the architecture design afterward.

THE LOGICAL OR CONCEPTUAL VIEW

The logical view is based on application domain entities necessary to implement the functional requirements.

The logical view specifies system decomposition into conceptual entities (such as objects) and connections between them (such as associations).

THE LOGICAL VIEW

The logical view is typically supported by

UML static diagrams including class/object diagrams and

UML dynamic diagrams, sequence diagram, state diagram.

THE DEVELOPMENT OR MODULE VIEW

The development view derives from the logical view and describes the static organization of the system modules.

UML diagrams such as package diagrams and component diagrams are often used to support this view.

THE PROCESS VIEW

The process view focuses on the dynamic aspects of the system, i.e., its execution time behavior.

This view maps functions, activities, and interactions onto runtime implementation.

The UML activity diagram support this view.

THE PHYSICAL VIEW

The physical view describes installation, configuration, and deployment of the software application.

It concerns itself with how to deliver the deploy-able system.

The physical view shows the mapping of software onto hardware.

The UML deployment diagrams are often used to support this view.

HAVE A GOO DAY!