Software Engineering(SE)

CSC 601

Subject Incharge

Varsha Nagpurkar

Assistant Professor

Room No. 407

email: varshanagpurkar@sfit.ac.in

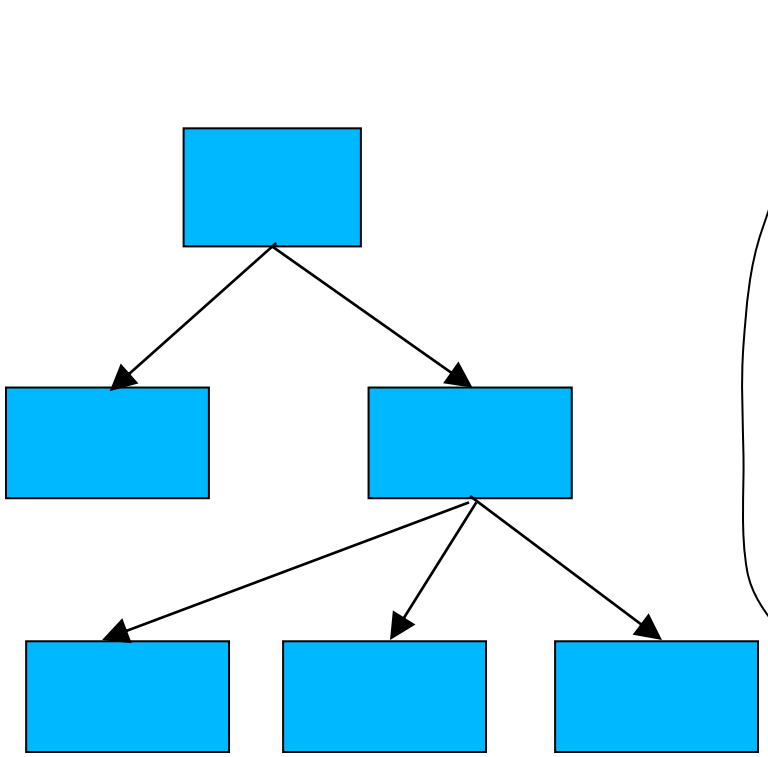
Module-4 Syllabus

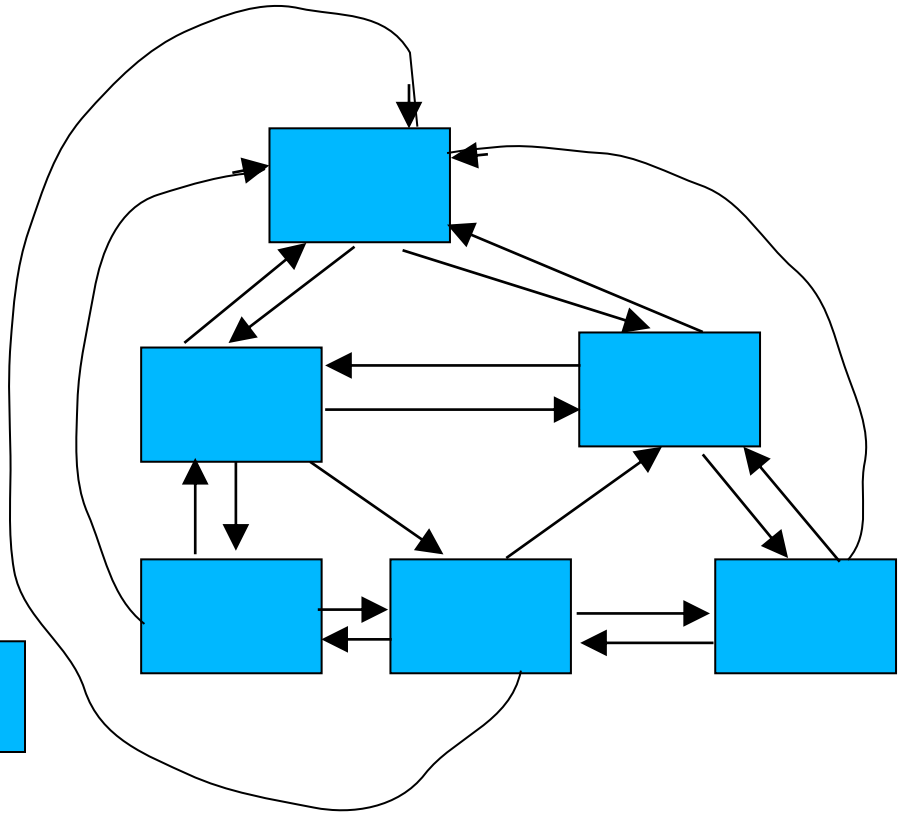
4.0		Software Design	10
	4.1	Design Principles, Design Concepts, Effective Modular Design – Cohesion and Coupling	
	4.2	Architectural Design	
	4.3	Component-level design	
	4.4	User Interface Design	

Modularity

- Modularity derives from the architecture.
- Modularity is a logical partitioning of the software design that allows complex software to be manageable for purposes of implementation and maintenance.
- The logic of partitioning is based on related functions, implementation considerations, data links, or other criteria.
- Modularity does imply interface overhead related to information exchange between modules and execution of modules.

Modularity





Modularity

- In technical terms, modules should display:
  - high cohesion
  - low coupling.

Cohesion

- It implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself
- Cohesion is the "single-mindedness" of a component
- Cohesion and coupling are necessary in making any software reliable and extendable

Cohesion and Coupling

- Cohesion is a measure of:
  - functional strength of a module.
  - A cohesive module performs a single task or function.
- Coupling between two modules:
  - a measure of the degree of interdependence or interaction between the two modules.

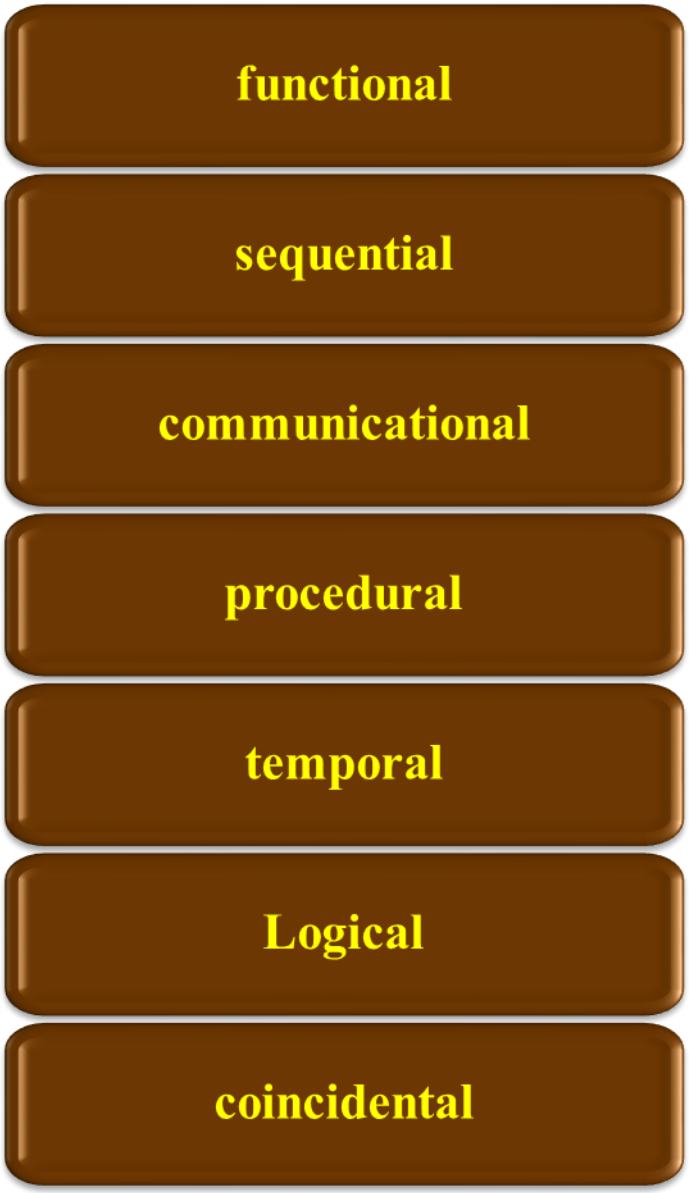
Cohesion and Coupling

A module having high cohesion and low coupling:

- functionally independent of other modules.
- Complexity of design is reduced
- an error existing in one module does not directly affect other modules.
- Reuse of modules is possible

```
 // Less cohesive class design

class BudgetReport {

    void connectToRDBMS() {

    }
    void generateBudgetReport() {
    }
    void saveToFile() {
    }
    void print() {
    }
}

More cohesive class design
// More cohesive class design
class BudgetReport {
    Options getReportingOptions() {
    }
    void generateBudgetReport(Options o) {
    }
}
class ConnectToRDBMS {
    DBconnection getRDBMS() {
    }
}
class PrintStuff {
    PrintOptions getPrintOptions() {
    }
}
class FileSaver {
    SaveOptions getFileSaveOptions() {
    }
}
```

Cohesion


functional
sequential
communicational
procedural
temporal
Logical
coincidental
Degree of cohesion

Functional cohesion

- Exhibited primarily by operations, this level of cohesion occurs when a module performs one and only one computation and then returns a result
- Different elements of a module cooperate:
  - to achieve a **single function**,
  - e.g. managing an employee's pay-roll.

Layer cohesion

- Exhibited by packages, components,and classes,it occurs when a higher layer accesses the services of a lower layer,but lower layers do not access higher layers
- For example, the Safehome security function requirement to make an outgoing phone call if an alarm is sensed

Layer Cohesion

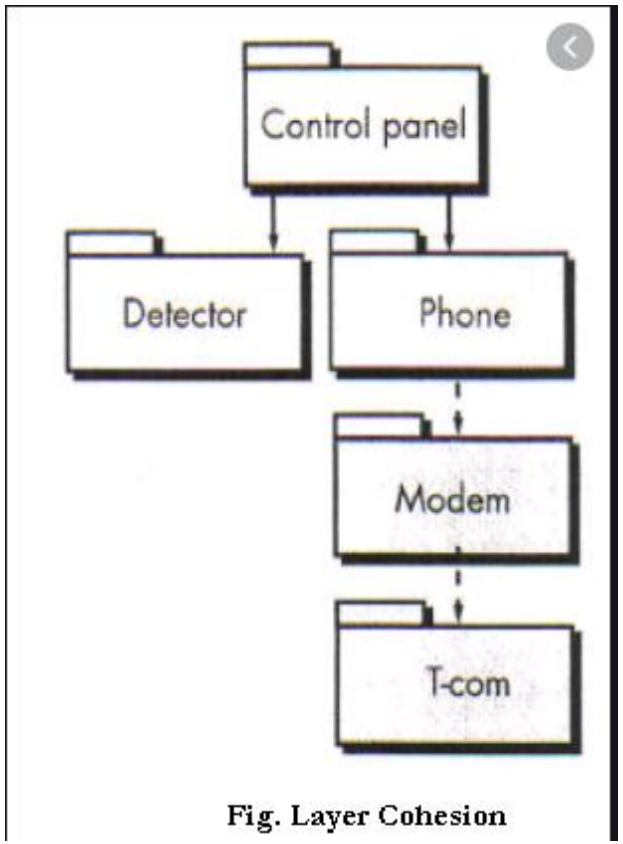


Fig. Layer Cohesion

Communicational cohesion

- All operations that access the same data are defined within one class
- All functions of the module:
  - reference or update the same data structure
  - Example:
  - Update record in data base and send it to the printer
    - Update a record on a database
    - Print the record

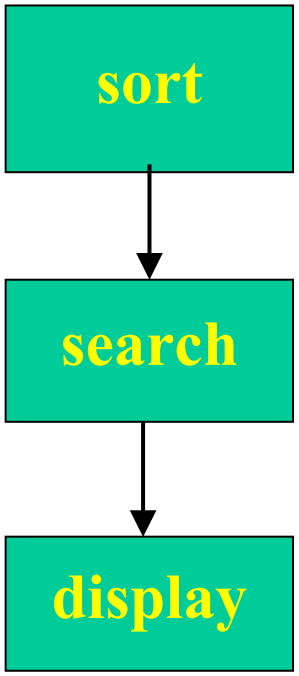
*

Cohesion

- Classes and components that exhibit functional,layer,and communicational cohesion are relatively easy to implement,test and maintain
- The designer should strive to achieve these levels of cohesion

Sequential cohesion

- Def: The output of one part is the input to another



Sequential cohesion

- Components or operations are grouped in a manner that allows the first to provide input to the next and so on
- The intent is to implement a sequence of operations

Procedural cohesion

- Def: Elements of a component are related only to ensure a particular order of execution
- The set of functions of the module:
  - certain sequence of steps have to be carried out in a certain order for achieving an objective,
  - E.g. Bank Transactions

*

Procedural cohesion

- Components or operations are grouped in a manner that allows one to be invoked immediately after the preceding one was invoked,even when there is no data passed between them

Temporal

- Operations that are performed to reflect a specific behaviour or state,
- e.g.,an operation performed at start-up or all operations performed when an error is detected

Temporal cohesion

Def: Elements are related by timing involved

Example:

- The set of functions responsible for
  - initialization,
  - start-up, shut-down of some process, etc.

Example: An exception handler that

- Closes all open files
- Creates an error log
- Notifies user

Example

- A system initialization routine: this routine contains all of the code for initializing all of the parts of the system.
- Lots of different activities occur, all at init time.

Logical cohesion

- All elements of the module perform similar operations:
  - e.g. **error handling, data input, data output, etc.**
- An example of logical cohesion:
  - a set of print functions or plot graphs to generate an output report arranged into a single module.

Example

- A component reads inputs from tape, disk, and network.
- All the code for these functions are in the same component.
- Operations are related, but the functions are significantly different.

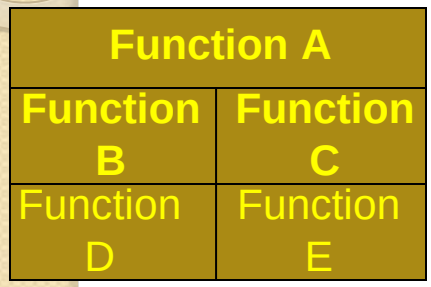
Utility cohesion

- Components,classes,or operations that exist within the same catogory but are otherwise unrelated are grouped together
- For example,a class called Statistics exhibits utility cohesion if it contains all attributes and operations required to compute six simple statistical measures

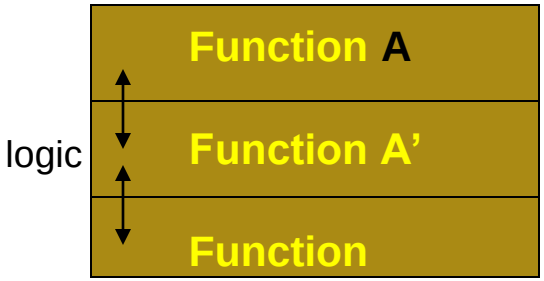
Cohesion

- These levels of cohesion are less desirable and should be avoided when design alternatives exist

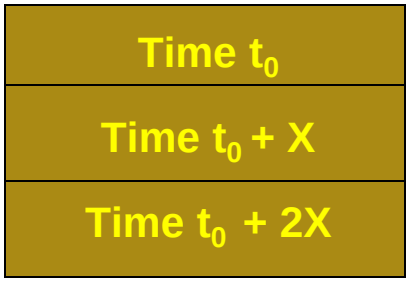
Examples of Cohesion



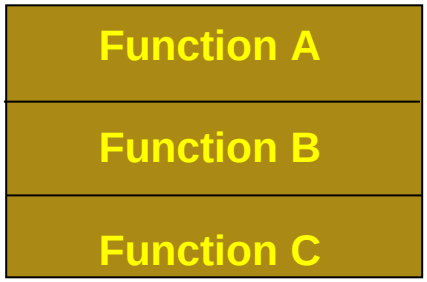
Coincidental
Parts unrelated



Logical
Similar functions

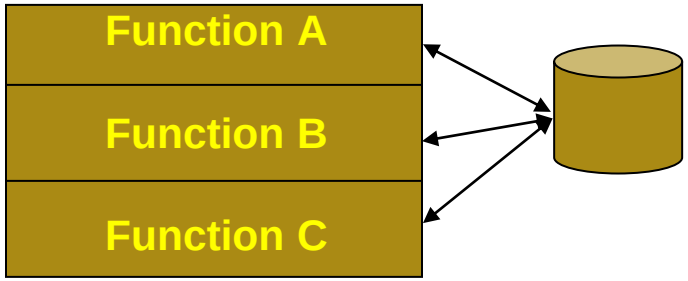


Temporal
Related by time

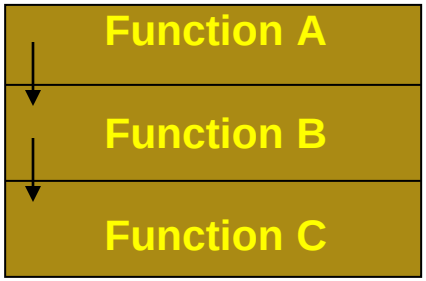


Procedural
Related by order of functions

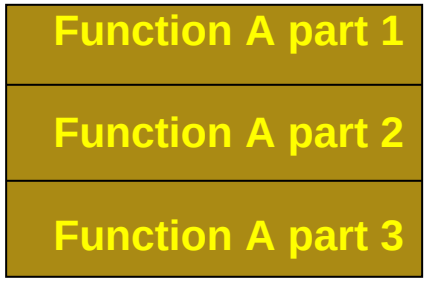
Examples of Cohesion (Cont.)



Communicational
Access same data



Sequential
Output of one is input to another



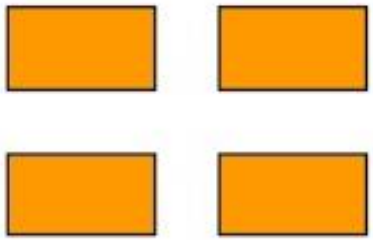
Functional
Sequential with complete, related functions

Coupling

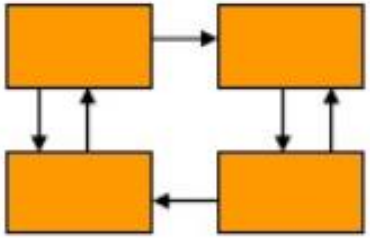
- Coupling indicates:
  - how closely two modules interact or how interdependent they are.
  - The degree of coupling between two modules depends on their interface complexity.

Coupling

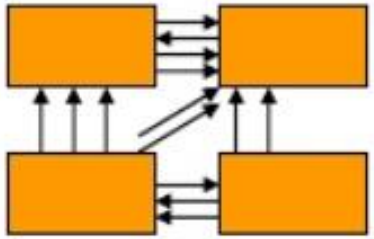
- Degree of dependence among components.



No dependencies



Loosely coupled-some dependencies



Highly couples-many dependencies

- High coupling makes modifying parts of the system difficult, e.g., modifying a component affects all the components to which the component is connected.

Coupling

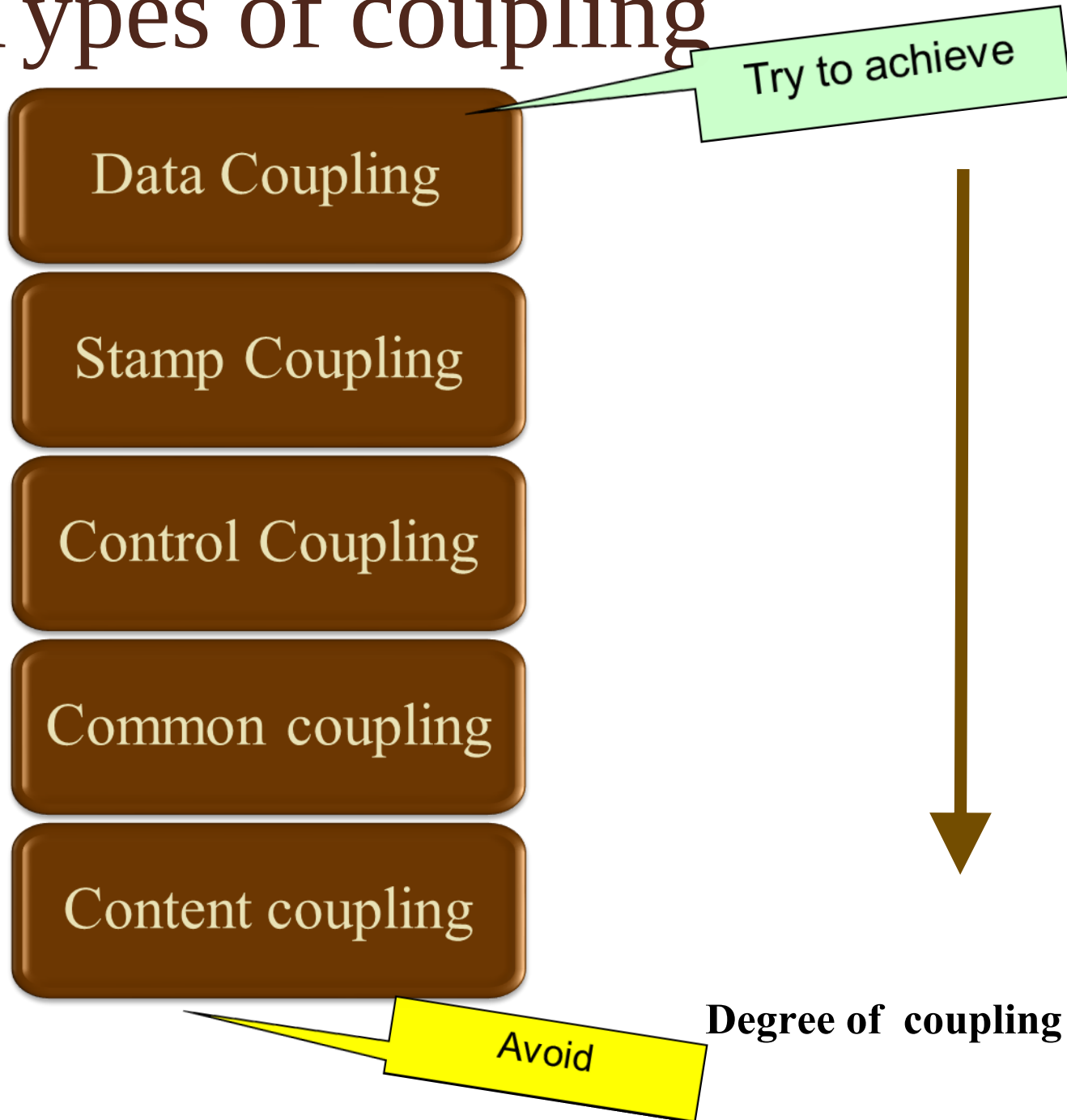
- Communication and collaboration are essential elements of any object-oriented system
- As the amount of communication and collaboration increases(i.e.,as the degree of "connectedness" between classes grows),the complexity of the system also increases
- And as complexity rises,the difficulty of implementing,testing,and maintaining

Coupling

- Coupling is a qualitative measure of the degree to which classes are connected to one another
- As classes(and components) become more dependent,coupling increases
- An important objective in component-level design is to keep coupling as low as is possible

*

Types of coupling



Content Coupling : (worst) When a module uses/alters data in another module

Common Coupling : 2 modules communicating via **global** data

External Coupling : Modules are tied to an environment external to the software

Control Coupling : 2 modules communicating with a control flag

Stamp Coupling : Communicating via a data structure **passed** as a parameter. The data structure holds **more** information than the recipient needs.

Data Coupling : (best) Communicating via parameter passing. The parameters passed are only those that the recipient needs.

No data coupling : independent modules.

Content coupling

- It occurs when one component "surreptitiously(in a secret or unauthorized way)modifies data that is internal to another component
- When a module alters/uses data in another module
- This violates information hiding-a basic design concept

Content coupling

- Def: One component modifies another
- Example:
  - Component directly modifies another's data
  - Component modifies another's code, e.g., jumps (goto) into the middle of a routine
- **The degree of coupling increases**
  - **from data coupling to content coupling**

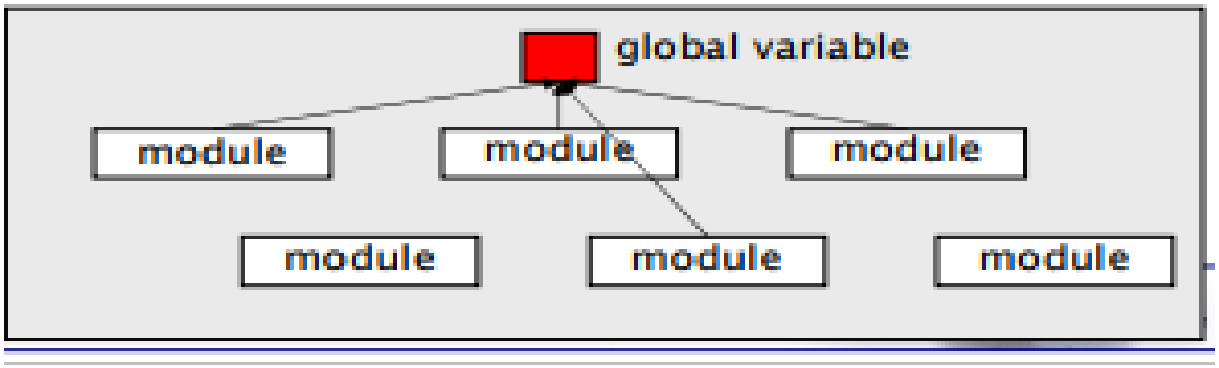
Common Coupling

- Occurs when a number of components all make use of a global variable
- Two modules communicating via global data
- Although this is sometimes necessary(e.g.for establishing default values that are applicable throughout an application),common coupling can lead to uncontrolled error propogation and unforeseen side effects when changes are made

Common Coupling

- Def: More than one component share data such as **global data structures**.

- Component passes control parameters to coupled components.



Common Coupling

- Two modules are common-coupled if they both have access to the same global variables

Drawbacks :

- It contradicts the spirit of structured programming
- Modules can have side effects
- Common coupled modules are difficult to reuse
- As a consequence of common coupling, a module may be exposed to more data than it needs

Control Coupling

- Occurs when operation A() invokes operation B() and passes a control flag to B
- Two modules communicating with the control flag
- The control flag then "directs" logical flow within B
- The problem with this form of coupling is that an unrelated change in B can result in the necessity to change the meaning of the control flag that A passes
- If this is overlooked,an error will result

Control coupling

- Data from one module is used to direct The order of instruction execution in another.
- Component passes control parameters to coupled components.
- Example 1: sort that takes a comparison function as an argument.
- Example 2: Main function controls all function
- Example 3: a flag set in one module and tested in another module

Stamp coupling

- Occurs when ClassB is declared as a type for an argument of an operation of Class A.
- Because ClassB is now a part of the definition of ClassA,modifying the system becomes more complex
- Communicating via a data structure passed as a parameter .The data structure holds more information than the recepient needs

Stamp coupling

- Def: Component passes a data structure to another component that does not have access to the entire structure.

Customer Billing System

- The print routine of the customer billing accepts customer data structure as an argument, parses it, and prints the name, address, and billing information
- Accessing date of birth to calculate age

Data coupling(Best)

- Occurs when operations pass long strings of data arguments
- The "bandwidth" of communication between classes and components grows and the complexity of the interface increases
- Testing and maintenance are more difficult
- Communicating via parameter passing
- The parameters passed are only those that the recepient needs

Data coupling

- Two modules are data coupled,
- if they communicate via a parameter:
  - an elementary data item,
  - Sharing of same data.
  - Withdraw , update and check balance
    Shares common data

Routine call coupling

- Occurs when one operation invokes another
- This level of coupling is common and is often quite necessary
- However,it does increase the connectedness of a system

Type use coupling

- Occurs when component A uses a data type defined in component B(e.g.,this occurs whenever “a class declares an instance variable or a local variable as having another class for its type”
- If the type definition changes,every component that uses the definition must also change

Inclusion or import

- Occurs when component A imports or includes a package or the content of component B

External

- Occurs when a component communicates or collaborates with infrastructure components(e.g.operating system functions,database capability,telecommunication functions)
- Although this type of coupling is necessary,it should be limited to a small number of components or classes within a system

COHESION	COUPLING
The measure of strength of the association of elements within a module	The measure of interdependence of one module to another
It is the degree to which the responsibility of a single component form a meaningful unit	It describes the relationship between software components
It is a property or characteristic of an individual module	It is a property of a collection of modules

Benefits of high cohesion and low coupling

High Cohesion

Cohesion refers to the measure of how strongly-related the functions of a module are. Low cohesion refers to modules that have different unrelated responsibilities. High cohesion refers to modules that have functions that are similar in many aspects.

The benefits of high cohesion are

- Readability – (closely) related functions are contained in a single module
- Maintainability – debugging tends to be contained in a single module
- Reusability – classes that have concentrated functionalities are not polluted with useless functions

Low Coupling

Coupling refers to the relationship of a module with another module. A module is said to be highly coupled with another module if changes to it will result to changes to the other module. And a module is said to be loosely coupled if a module is independent of any other modules. This can be achieved by having a stable interface that effectively hides the implementation of another module.

~~Benefits of low coupling are~~

- maintainability – changes are confined in a single module
- testability – modules involved in unit testing can be limited to a minimum
- readability – classes that need to be analyzed are kept at a minimum

University Questions

- Explain coupling and cohesion?Explain the types of cohesion with examples.(10 marks-Dec-19)
- Explain coupling and cohesion?Explain the types of coupling with examples.(10 marks-May-19)

*

User Interface Design

User Interface Design

- User interface is considered as a front-end application view which is interacted by the user so as to use the software
- User is able to manipulate as well as control the software and hardware with the help of user interface
- Now-a-day,user interface appears in each and every place where there is existence of digital technology,right from desktops,smart phones,cars,music players,airplanes,ships etc.

User Interface Design

- User interface is an integral component of software and is designed in such a manner that it should be able to provide the user insight of the software
- UI offers a fundamental platform for the communication of user and computer
- The form of UI can be graphical,text-based,audio-video based.It is usually depends upon the undelying platform(hardware and software combination

User Interface Design

The popularity of software depends upon following aspects

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interfacing screens

Categories of UI

- Command Line Interface
- Graphical User Interface

Command Line Interface

- In this approach,user communicates with computer system with the help of commands
- In Unix/Linux system the command interpreter is known as shell.
- Shell is nothing but the interface between user and operating system

Graphical User Interface(GUI)

- Another approach to interface with the computer system is through a user friendly graphical user interface or GUI
- Instead of directly entering commands through a command-line interface,a GUI allows a mouse-based,window-and-menu based system as an interface

Characteristics of Good User Interface

- Clarity
- Conclusion
- Familiarity
- Responsiveness
- Consistency
- Aesthetics
- Efficiency
- Attractive
- Forgiveness

Benefits of Good Interface Design

- Higher revenue
- Increased user efficiency and satisfaction
- Reduced development costs
- Reduced support costs

The Golden Rules

- The following 3 rules form the basis for a set of user interface design principles and guidelines
  - Place the user in control
  - Reduce the user's memory load
  - Make the interface consistent

Mandel defines a number of design principles that allow the user to maintain control

- **Define interaction modes in a way that does not force a user into unnecessary or undesired actions-**
- An interaction mode is the current state of the interface
- For example,if spell check is selected in a word-processor menu, the software moves to a spell-checking mode
- There is no reason to force the user to remain in spell-checking mode if the user desires to make a small text edit along the way
- The user should be able to enter and exit the mode with little or no effort

Mandel defines a number of design principles that allow the user to maintain control

- **Provide for flexible interaction.-**
- Because different users have different interaction preferences, choices should be provided. For example, software might allow a user to interact via keyboard commands, mouse movement, a digitizer pen, a multitouch screen, or voice recognition commands. But every action is not amenable to every interaction mechanism. Consider, for example, the difftculty of using keyboard command (or voice input) to draw a complex shape.

Mandel defines a number of design principles that allow the user to maintain control

- **Allow user interaction to be interruptible and undoable.-**
- Even when in- volved in a sequence of actions, the user should be able to interrupt the sequence to do something else (without losing the work that had been done). The user should also be able to "undo" any action.

Mandel defines a number of design principles that allow the user to maintain control

- **Streamline interaction as skill levels advance and allow the interaction to be customized.-**
- Users often find that they perform the same sequence of interactions repeatedly. It is worthwhile to design a "macro" mechanism that enables an advanced user to customize the interface to facilitate interaction.

Mandel defines a number of design principles that allow the user to maintain control

- **Hide technical internals from the casual user-**
- The user interface should move the user into the virtual world of the application.
- The user should not be aware of the operating system, file management functions, or other arcane computing technology.

Mandel defines a number of design principles that allow the user to maintain control

- **Design for direct interaction with objects that appear on the screen-**
- The user feels a sense of control when able to manipulate the objects that are necessary to perform a task in a manner similar to what would occur if the object were a physical thing.
- For example, an application interface that allows a user to drag a document into the "trash" is an implementation of direct manipulation.

Reduce the User's Memory Load

- **Reduce demand on short-term memory-**
- When users are involved in complex tasks, the demand on short-term memory can be signiftcant. The interface should be designed to reduce the requirement to remember past actions, inputs, and results. This can be accomplished by providing visual cues that enable a user to recognize past actions, rather than having to recall them

Reduce the User's Memory Load

- **Establish meaningful defaults.** The initial set of defaults should make sense for the average user, but a user should be able to specify individual preferences. However, a "reset" option should be available, enabling the redeftnition of original default values.

Reduce the User's Memory Load

- **Define shortcuts that are intuitive-** When mnemonics are used to accomplish a system function (e.g., alt-P to invoke the *print* function), the mnemonic should be tied to the action in a way that is easy to remember (e.g., first letter of the task to be invoked).

*

Reduce the User's Memory Load

- **The visual layout of the interface should be based on a real-world met- aphor.** For example, a bill payment system should use a checkbook and check register metaphor to guide the user through the bill paying process. This enables the user to rely on well-understood visual cues, rather than memorizing an arcane interaction sequence.

Reduce the User's Memory Load

- **Disclose information in a progressive fashion-**
- The interface should be organized hierarchically. That is, information about a task, an object, or some behavior should be presented first at a high level of abstraction. More detail should be presented after the user indicates interest.

Make the Interface Consistent

- **Allow the user to put the current task into a meaningful context-**
- Many interfaces implement complex layers of interactions with dozens of screen images. It is important to provide indicators (e.g., window titles, graphical icons, consistent color coding) that enable the user to know the context of the work at hand. In addition, the user should be able to determine where he has come from and what alternatives exist for a transition to a new task.

Make the Interface Consistent

- **Maintain consistency across a complete product line-**
- A family of applica- tions (i.e., a product line) should implement the same design rules so that consistency is maintained for all interaction.

Make the Interface Consistent

- **If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so-**
- Once a particular interactive sequence has become a de facto standard (e.g., the use of alt-S to save a ftle), the user expects this in every application encountered. A change (e.g., using alt-S to invoke scaling) will cause confusion.