Lecture 06: Software Design

Software Engineering (CSM 31212)

MM. Mohamed Mufassirin
Lecturer in Computer Science
Dept. of Computer Science
FAS/SEUSL

Software Design

Learning Outcomes

- Understand the activities involved in the Design process
- Understand the principles of good design
- Identify Software design techniques
- Identify possible software architectures

Software Design - Objectives

- Design is a meaningful engineering representation of something that is to be built.
- In software Engineering context, design focuses on transforming requirements into implementable version of the software system.

Design Activities

- Identification of the sub-systems
- Identification of the software components
- Identification of the software architecture
- Data design
- Interface design
- Algorithm design
- Data structure design
- Design specification

Design Process

Step 1: Understanding the SRS/ Problem

Step 2: Produce initial design

Step 3: Validate it

if valid, DONE

else refine it and revalidate

Step 4: Iterate Step 3 until obtain a good design

Software Design – Why is it important?

- A good design is the key for a successful software system
- A good design allows easy maintenance of a system
- A good design allows to achieve non-functional requirements such as reliability, performance, reusability, portability.
- A good design facilitates the development and management processes of a software project.

Some Important Software Design Principles

- Abstraction
- Modularity
- Information Hiding (Encapsulation)
- Polymorphism

Abstraction

- This is an intellectual tool (a psychological notion) which permits one to <u>concentrate on a problem at some level of generalization without regard to irrelevant low level details</u>.
- Abstraction allows us to proceed with the development work without been held up in low-level implementation details (yet to be discovered)

Eg: Online Shopping App – Payment Methods

Modularity

- Software is divided into separately named, addressable components called modules.
- Complexity of a program depends on modularity

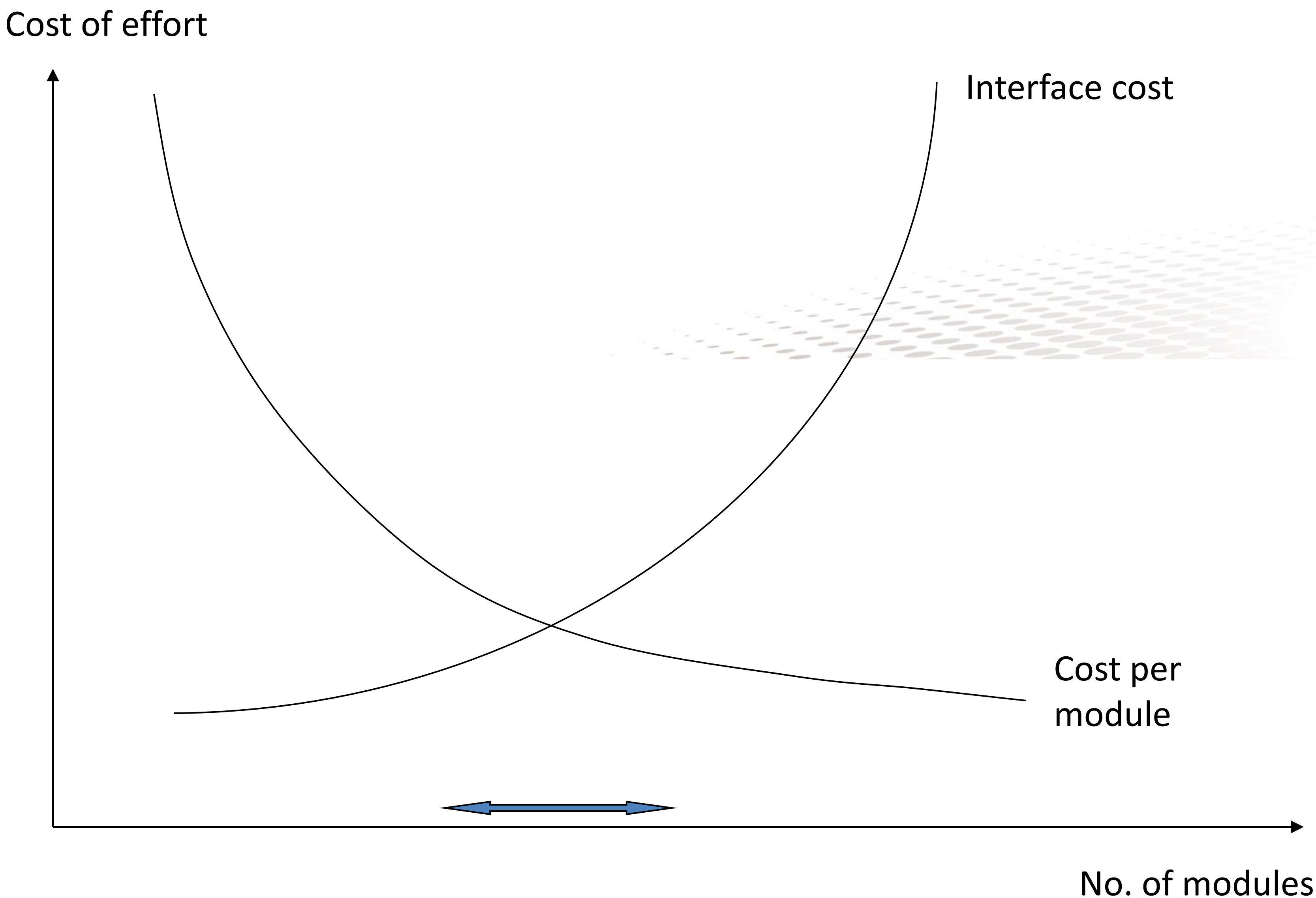
Modularity   Cont...

Modularity Facilitates

- the development process
- the maintenance process
- the project management process
- reusability

Eg: A restaurant management system (Billing, Inventory, Reservation)

How Many Modules?



Module Coupling

- A measure of the strength of the interconnections between system components.
- **Loose (Low) coupling** means component changes are likely to affect other components.
- Shared variables or control information exchange lead to **tight (High) coupling**.
- Loose coupling can be achieved by component communication via parameters or message passing.

Levels of Coupling

- **Data Coupling**

  Data is passed from one module to another using arguments

- **Stamp Coupling**

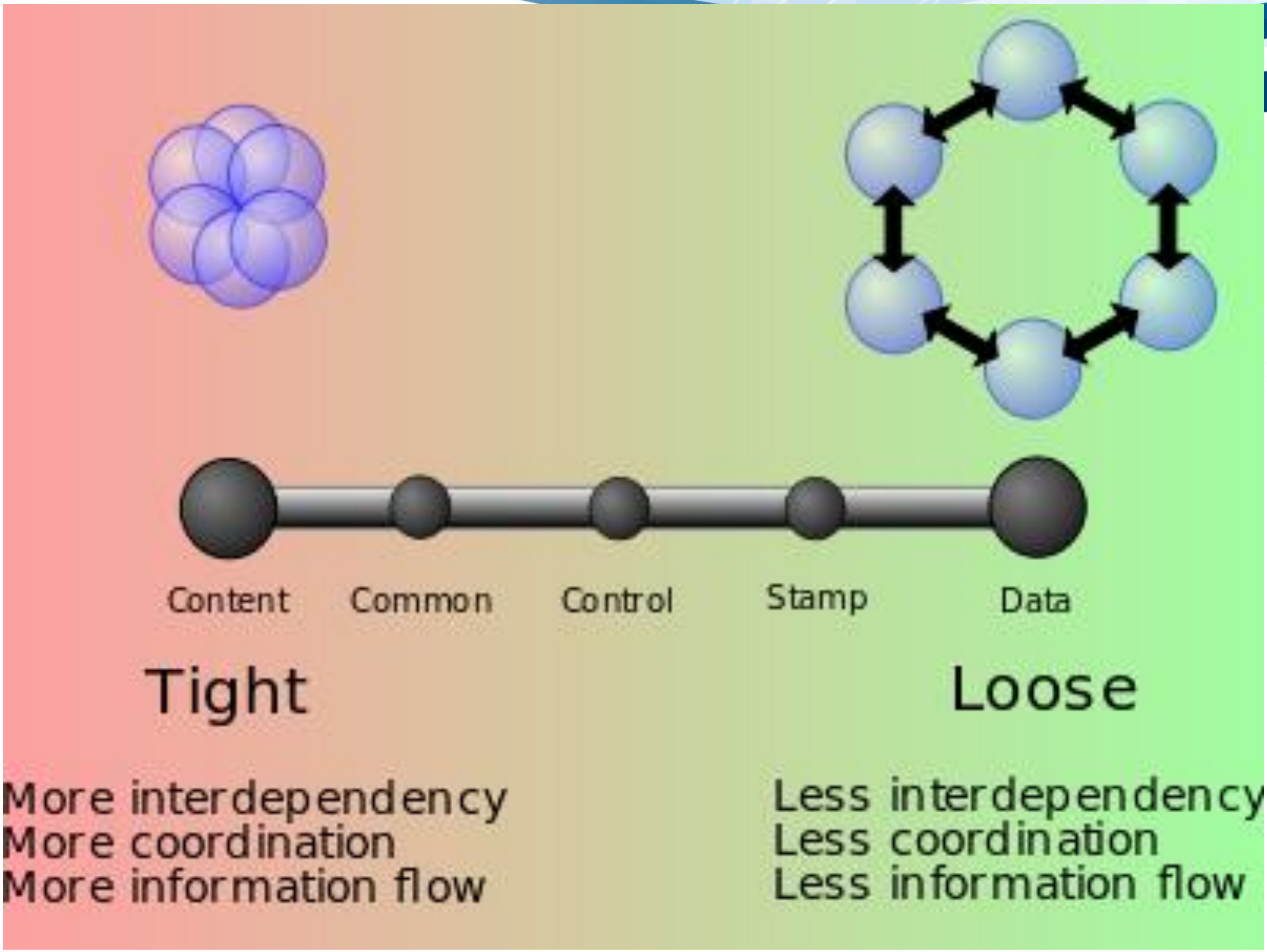
  More data than necessary is passed via arguments. Eg. Pass the whole record instead of just the field being changed.

- **Control Coupling**

  A flag is passed from one module to another affecting the functionality of the second module

- **External Coupling**

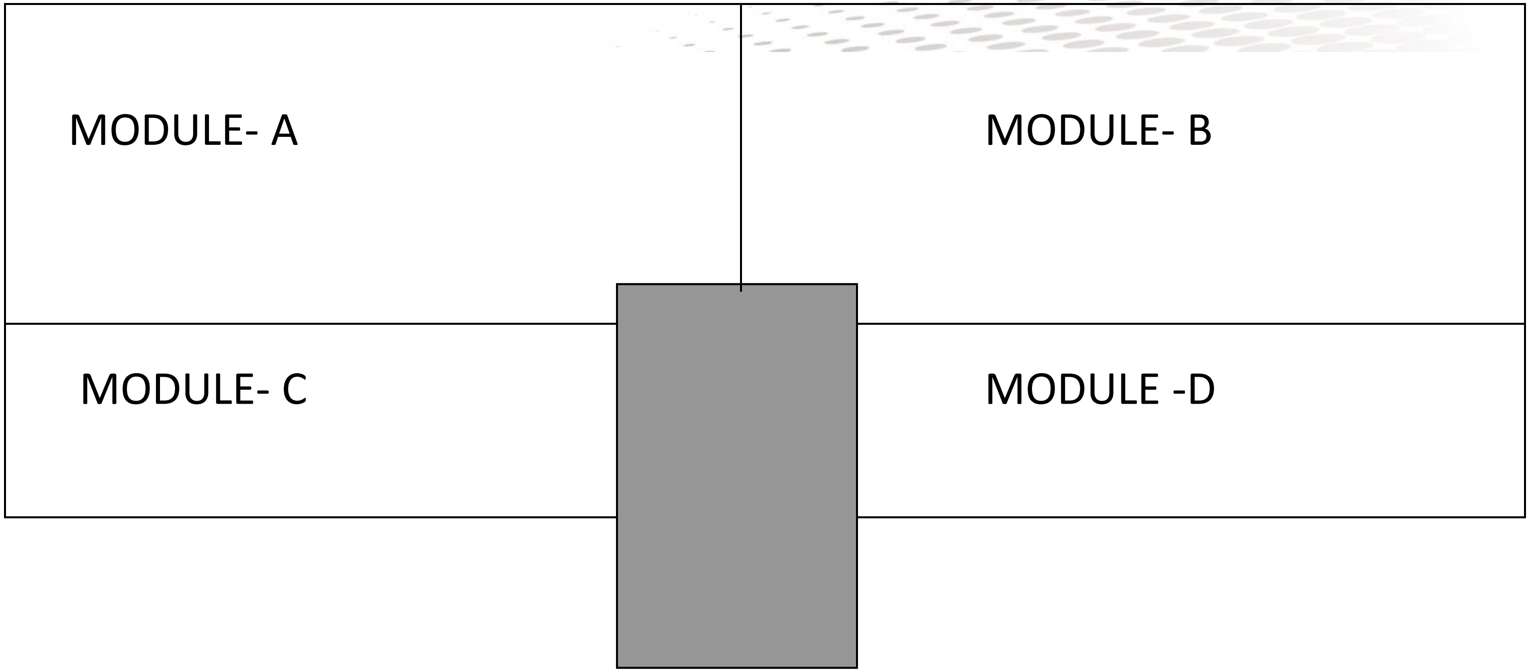
  Coupling with the environment (eg. Data files, other programs etc.).


Content
Common
Control
Stamp
Data
Tight
Loose
More interdependency
More coordination
More information flow
Less interdependency
Less coordination
Less information flow

Levels of Coupling (Cont..)

- **Common Coupling**

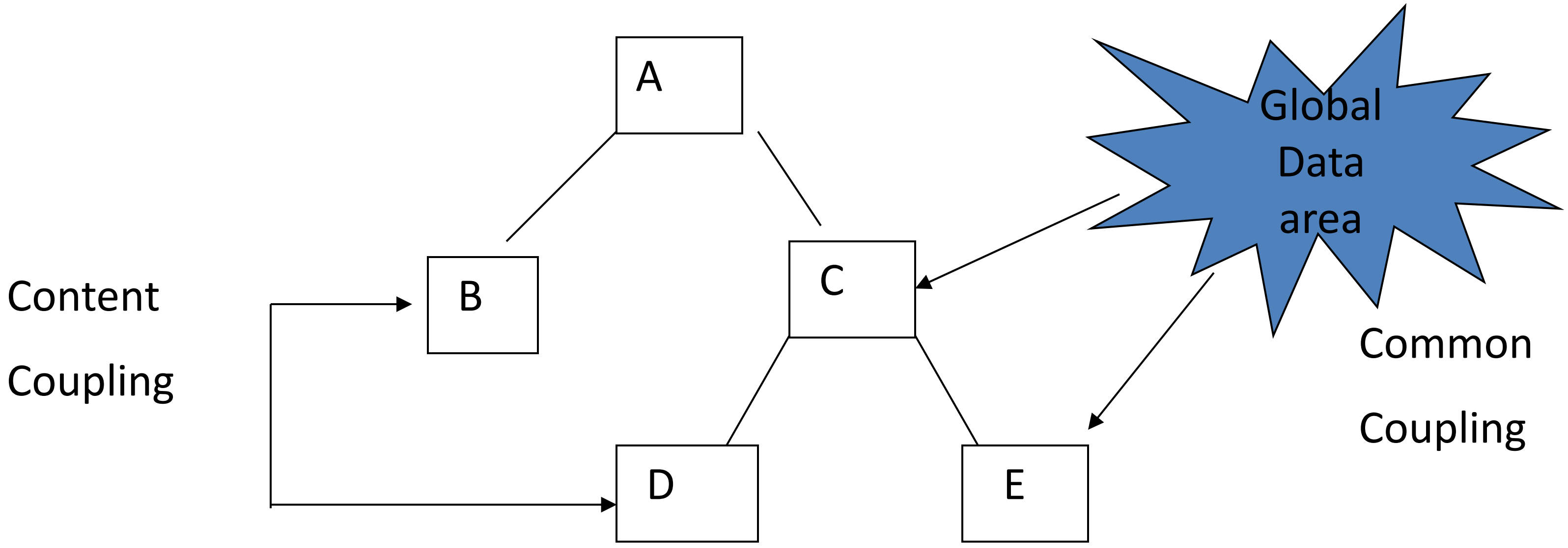
Occurs when modules access the same global data.



Levels of Coupling (Cont..)

- **Content Coupling**

One module directly affects the working of another. Calling module can modify the called module or refer to an internally defined data element.

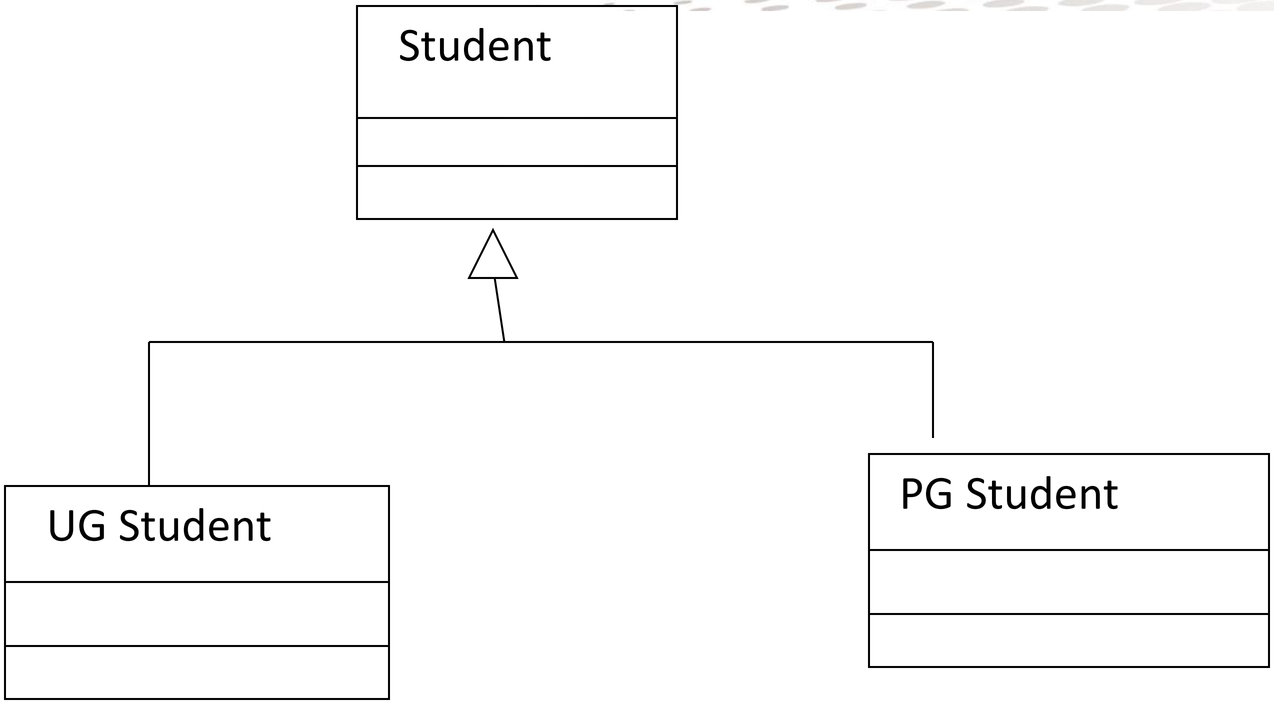


Levels of Coupling (Contd..)

Object oriented systems are loosely coupled. No shared state and objects communicate using message passing,

- **Object Coupling**

Occurs when an object class inherits attributes and methods of another class. Changes to super-class propagate to all sub-classes.



Coupling should be minimized.

Loosely coupled modules facilitate:

- Maintenance
- Development
- Reusability

Information Hiding (Encapsulation)

- Encapsulation is a technique for minimizing interdependencies among separately written modules by defining strict external interfaces.
- The external interface acts as a contract between a module and its clients. If clients only depend on the interface, modules can be re-implemented without affecting the client. Thus, the effects of changes can be confined.

Design with Reuse

✓ Another important design consideration is reusability of software components. Designing reusable software components is extremely valuable.

Some benefits of software reuse are:

- Increased reliability
- Reduced process risks
- Effective use of specialists
- Standards compliance
- Accelerated Development

Software Architectural Design

The architectural design process is concerned with establishing a basic structured framework for a system. It involves identifying the major components of the system and the communications between these components.

Large systems are always decomposed into subsystems that provide some related set of services. The initial design process of identifying these sub-systems and establishing a framework for sub-system control and communication is called *architectural design* and the output of this design process is a description of the *software architecture.*

Sub-systems and Components

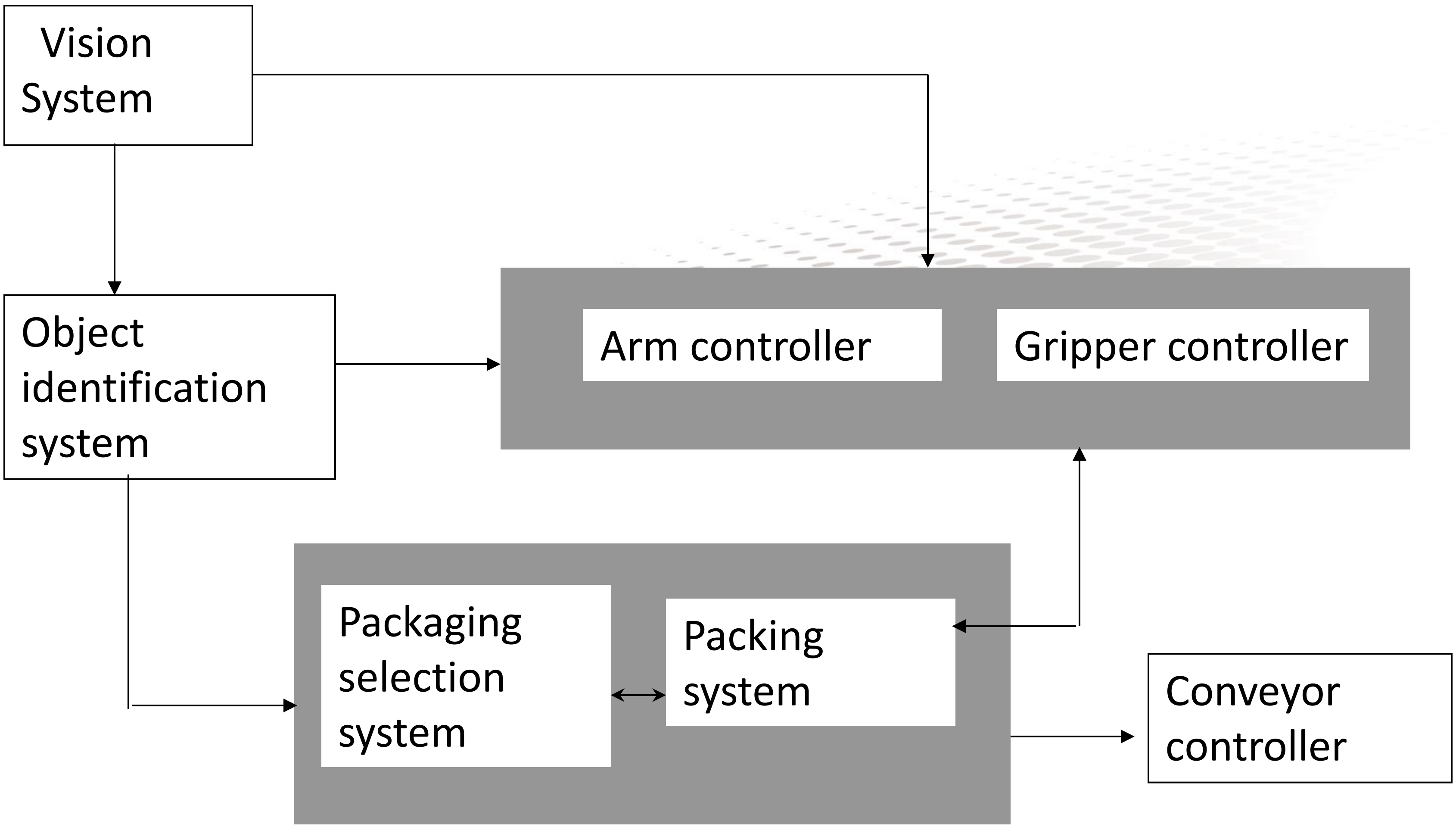
Sub- Systems - A Sub-system is a system in its own right whose operations are not depend on the services proved by the other sub-systems. Sub systems are composed of modules (components) and have defined interfaces which are used for communication with other sub-systems,.

Components – A component (module) is normally a system component that provides one or more services to other modules. It makes use of services provided by other modules. It is not normally considered as an independent system. Modules are usually composed from a number of other, simpler system components.

Architectural Design Process

- **System structuring** - The system is structured into a number of principal sub-systems where a sub-system is an independent software unit. Communications between sub-systems are identified.
- **Control modelling** – A general model of the control relationships between the parts of the systems is established.
- **Modular decomposition** - Each identified sub-system is decomposed into modules. The architect must decide on the types of module and their interconnections.

Architectural Design – An example



Repository Model

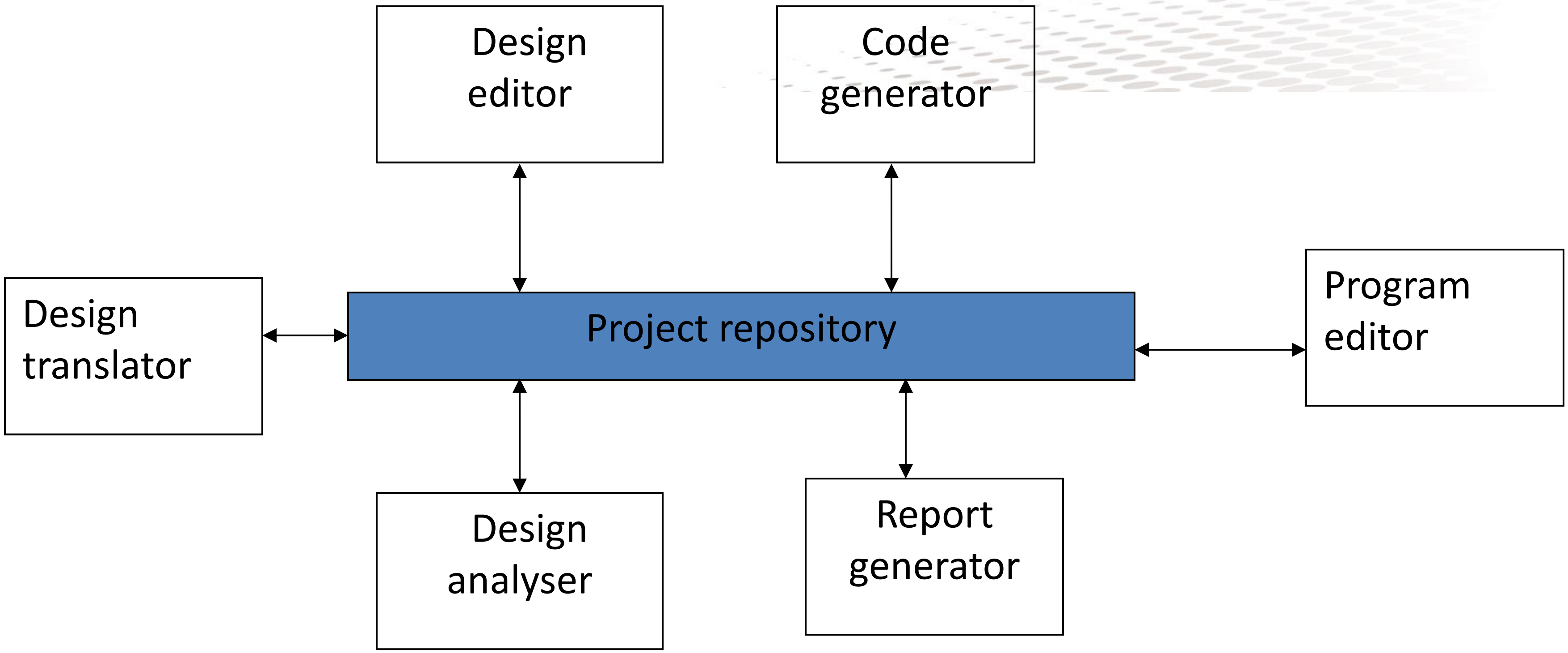
Sub-systems making a up a system must exchange information so that they can work together effectively. One approach is to keep all **shared data in a central database** that can be accessed by all sub-systems. A system model based on a shared database is called *repository model*.

This model is suited to applications where data is generated by one sub-system and used by another.

Examples of this type of systems include command and control systems, management information systems CAD systems and CASE tools.

Repository Model - An example

The architecture of an integrated CASE tool.



Some advantages of Repository Model

- It is efficient way to share large amount of data. There is no need to transmit data explicitly from one sub-system to another.

- Activities such as backup recovery, access control and recovery from error are centralized. They are the responsibility of the repository manager. Tools can focus on their principal function rather than be concerned with these issue.

- The model of sharing is visible through the repository schema. It is straight forward to integrate new tools given that they are compatible with the agreed data model.

Some disadvantages of Repository model

- Sub-systems must agree on a repository data model. Certainly, this is a compromise between the specific needs of each tool. Performance may be adversely affected by this compromise. It may be difficult or impossible to integrate new sub-systems if their data models do not fit the agreed schema.

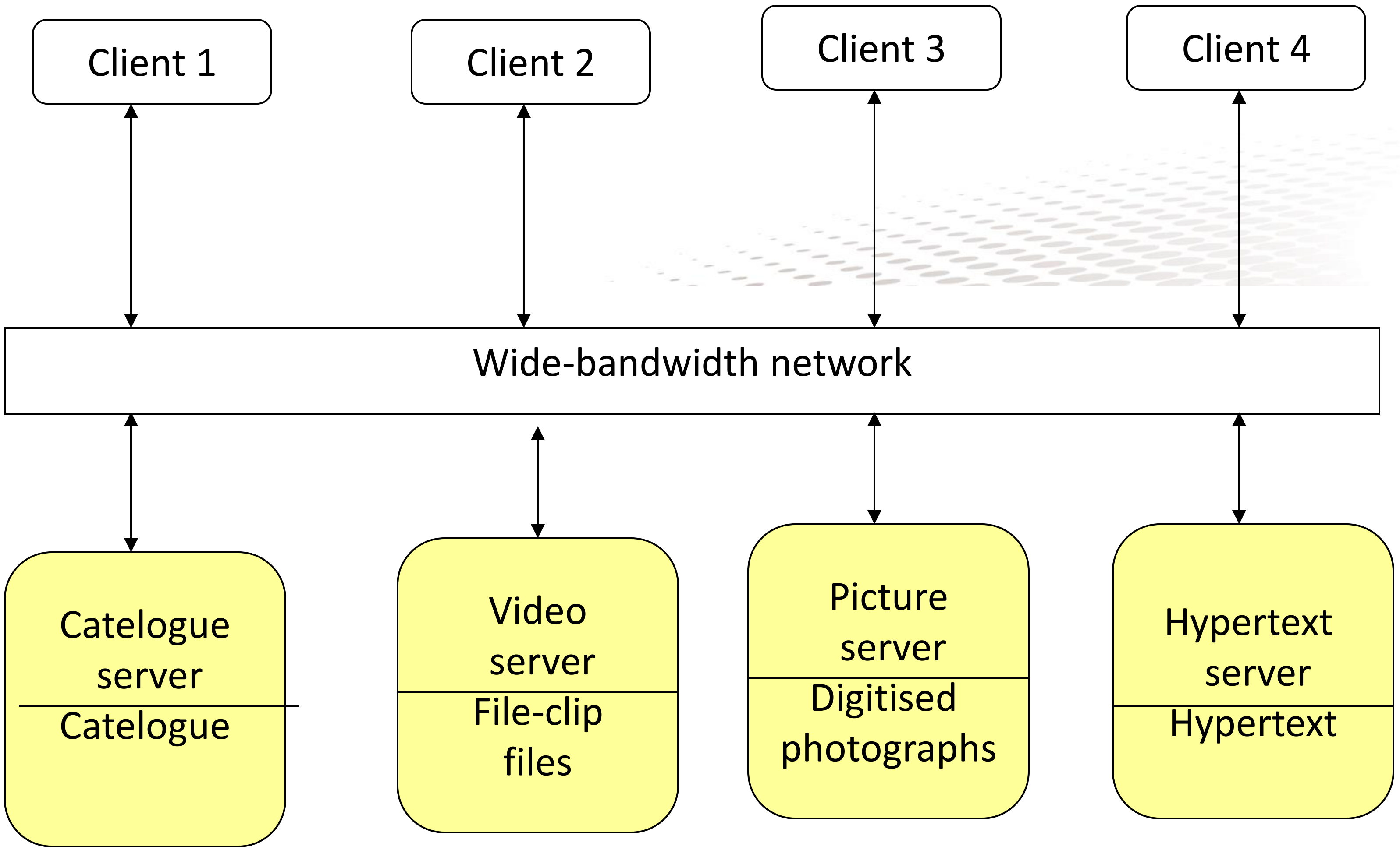
- Evolution may be difficult as a large volume of information is generated according to an agreed data model. Translating this to a new model will certainly be expensive.

The Client-Server Model

- The client-server architectural model is a distributed system model which shows how data and processing are distributed across a range of processors. The major components of the model are:

1. A set of stand-alone servers which offer services to other sub-systems.
   - Examples of servers are print servers, web servers and data base servers.
2. A set of clients that call on the services offered by the servers. Theses are normally sub-systems in their own right. There may be several instances of a client program executing concurrently.
3. A network which allows the clients to access these services.

Client- Server model - An example



User Interface Design

Good user interface design is critical to the success of a system. An interface that is difficult to use will, at best, result in a high level of user errors. At worst, user users will simply refuse to use the software system irrespective of its functionality.

If information is presented in a confusing or misleading way, users may misunderstand the meaning of information. They may initiate a sequence of actions that corrupt data or even cause catastrophic system failure.

The system should assist the user providing help facilities and should guide the user in the case of occurrence of an error.

Graphical User Interfaces

- Although text-based interfaces are still widely used, especially in legacy systems, computer users now expect application systems to have some form of graphical user interface.

The advantages of GUI are:

1. They are relatively easy to learn and use. Users with no computing experience can learn to use the interface after a brief training session.

2. The users have multiple screens (windows) for system interaction. Switching from one task to another is possible without losing sight of information generated during the first task.

3. Fast, full-screen interaction is possible with immediate access to anywhere on the screen.

The Characteristics of a Graphical User Interface

Windows - Multiple windows allow different information to be displayed simultaneously on the user's screen

Icons - Icons represent different types of information. On some system icons represents files, on others, icon represents 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 interest in a window.

Graphics - Graphical elements can be mixed with text on the same display.

User Interface Design Principles

User familiarity - The interface should use terms and concepts which are drawn from the experience of the people who will make most use of the system.

Consistency – The interface should be consistent in that, wherever possible, comparable operations should be activated in the same way.

Recoverability – The interface should include mechanisms to allow users to recover from errors.

User guidance – The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities.

User diversity – The interface should provide appropriate interaction facilities for different type of system user.

Colour in Interface Design

Some guidelines for effective use of colour in user interfaces.

1. You should not use more than four or five separate colours in a window and no more than seven in a system interface. Colours should be used selectively and consistently.

2. Use colour change to show a change in system status. If the display changes colour, this should mean that a significant event has occurred.

3. Use colour coding to support the task which users are trying to perform. If they have to identify anomalous instances, highlight these instances.

Colour in Interface Design (Cont…)

4. Be careful about colour pairing. Some colour combinations are not good for the eye. (eg. Red and Blue)

5. Use colour coding in a useful and consistent way. If one part of a system displays error messages in red, then red should not be used for anything else.

Software Design Techniques

- Top-Down Decomposition
- Bottom –Up Design
- Jackson Structured Design
- Object Oriented Design