Tribhuvan University

Institute of Engineering

Pulchowk Campus

Department of Electronics and Computer Engineering

Software Engineering

Chapter Three ,Five,Six

Architecture Design,CBSE,Software Reuse

by

Er. Bishwas Pokharel

Lecturer, Kathford Int'l College of Engg. and Mgmt.

Date: 19th jan, 2018

Software Design

IEEE defines software design as "both a process of defining the architecture, components, interfaces, and other characteristics of a system or component and the result of that process."

Design creates a representation or model of the software, but unlike the requirements model (that focuses on describing required data, function, and behavior), the design model provides detail about software architecture, data structures, interfaces, and components that are necessary to implement the system.

CBSE(computer based software engineering)

Interface: An interface is a shared boundary across which information is passed. Interfaces apply to hardware, software and human interaction

- **Data communications interfaces.**

  The passing of information between systems by data communications transport mechanisms, protocols(HTTP,SMTP) and network architectures
- **Human machine interfaces (user interface).** The means by which human beings interact with the system
- **Electrical interfaces.** The means by which systems are interconnected by electrical cabling.

A system is a generic term used to describe a mechanism/technology/design/... that works to serve a set of functions. **A component** is a sub-system.

Ex:

- A car is a system. This system has components such as engine, seating, and dashboard. A car is an object and some of its components are also objects.
- Every software program/application is a system and have different components. From the perspective of a UX designer, the components of a software system can be its navigation, different sections, forms, and screens.

- From the perspective of a software architect, this system has main components such as frontend and various backend services.
- From the perspective of a developer, this system has different components such as libraries and classes.
- In programming and engineering disciplines, a component is an identifiable part of a larger program or construction. Usually, a component provides a particular function or group of related functions.
- components communicate with each other via *interfaces*

Component-based software development

- Component-based software development is a new trend in software development.
- The main idea is to reuse already completed components instead of developing everything from the very beginning each time.
- Earliest cases of successful reuse is the development of different libraries, EX: mathematical libraries. These libraries include functions (Ex: mathematical functions such as sine, cosine, matrix operations, etc.), which are referred to in the source code and then linked together with the proprietary code.

The success of this type of reusable entities lies in several facts:

- There exists well-defined theory about these types of functions.
- The communication between the application and these functions is simple. It is of procedural type. The application invokes the functions sending to it input parameters, and the library responds by the execution of the function and returning the output parameters.
- The inputs and outputs are precisely defined

- Relative good error handling – If the inputs are erroneous, the output will usually return a specific value denoting an error.

A disadvantage and limitation of these types of components is Inflexibility

- For a new version of the library, the application must be rebuilt
- The limitations of types of input and output parameters. When changing types of parameters (for example using text elements instead of numbers in a sort function), a new library function must be used.

- Reuse of software components concept has been taken from manufacturing industry and civil engineering field. Manufacturing of vehicles from parts and construction of buildings from bricks are the examples.
- Car manufacturers have not been so successful if they have not used standardized parts/components.
- Software companies have used the same concept to develop software in standardize parts/components. Software components are shipped with the libraries available with software.

- Microsoft Corporation and Sun Microsystems are two major software-providing organizations.
- These companies have provided parts/components with their software to market themselves successful and their tools are widely used in software industry.
- For example, Microsoft Visual Basic (VB), JBuilder, Microsoft.Net, Power Builder, Delphi, etc provides an IDE (Integrated Development Environment).
- IDE helps programmers to develop programs in such a short time which is only possible using an IDE.

- As the name indicates, IDE provides an environment in which components are available in the toolbox or in the reference library like a car assembling plant.
- We do not need to develop the components during the assembling of the car but they are there and we timely assemble them.
- Similarly in IDE, the standard components such as text box, label box and command button are available in the toolbox and we just integrate and use them.

- Visual Basic also allows executing a program without compilation in order to just see the results.

- We do not need separate programs to develop the components or to run its program. This is the reasons VB is most widely used Rapid Application Development Tool (RAD).

- Vendors other than Microsoft and Sun Microsystems are also providing components with their tools to make them successful.

- Different people have defined the component in different ways. The most popular definition is "Each reusable binary piece of code is called a component"
- The component concept is similar to object concept of object-oriented programming. A component is an independent part of the system having complete functionalities. A component is designed to solve a particular purpose, such as command button and text box of VB The component is like a pattern that forces the developers to use the predefined procedures and meets the specifications to plug it into the new SW components.

What are components?

1. A component is a non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces.

2. A run-time software component is a dynamically bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at run-time.

3. The software component definition is widely accepted today, which says that a component is a part of software in a binary form (i.e. it is not necessary to rebuilt it), with contractually specified interfaces (i.e. defined API and all assumptions in which the component can work), A component can be deployed independently (i.e. it can be dynamically loaded into the system, or dynamically replaced). It is a subject to composition by third party (i.e a component must have a mechanism which makes it possible to integrated it in the system without modifying and rebuilding it)

Component characteristics 1

Standardised	Component standardisation means that a component that is used in a CBSE process has to conform to some standardised component model. This model may define component interfaces, component meta-data, documentation, composition and deployment.
Independent	A component should be independent Š it should be possible to compose and deploy it without having to use other specific components. In situations where the component needs externally provided services, these should be explicitly set out in a ŌrequiresÕ interface specification.
Composable	For a component to be composable, all external interactions must take place through publicly defined interfaces. In addition, it must provide external access to information about itself such as its methods and attributes.

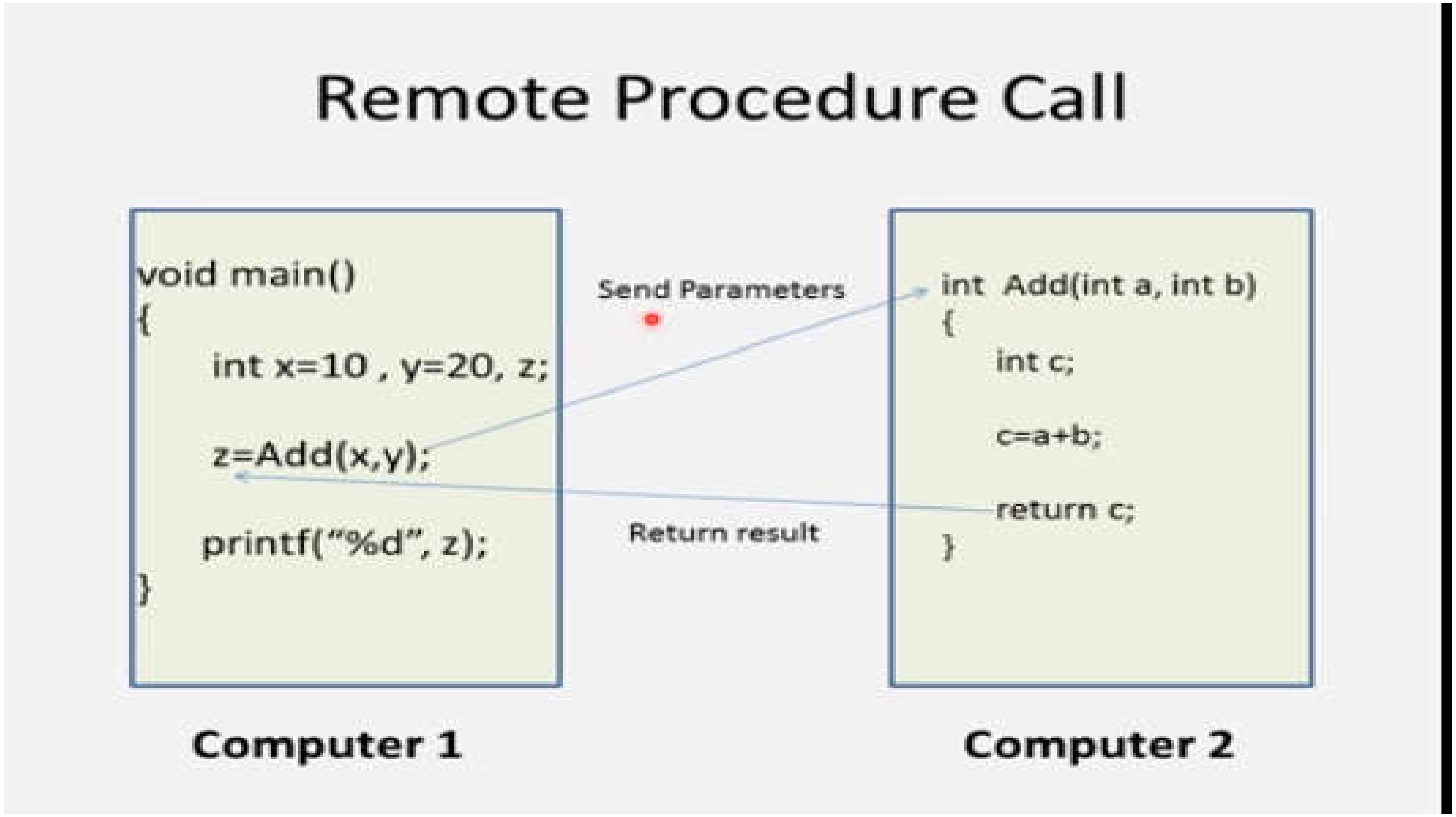
Component characteristics 2

Deployable	To be deployable, a component has to be self-contained and must be able to operate as a stand-alone entity on some component platform that implements the component model. This usually means that the component is a binary component that does not have to be compiled before it is deployed.
Documented	Components have to be fully documented so that potential users of the component can decide whether or not they meet their needs. The syntax and, ideally, the semantics of all component interfaces have to be specified.

Component as a service provider

- The component is an independent, executable entity.
- It does not have to be compiled before it is used with other components.
- The services offered by a component are made available through an interface and all component interactions take place through that interface.

Middleware Technologies



Remote Procedure Call

Computer 1

```
void main()
{
        int x=10 , y=20, z;

        z=Add(x,y);

        printf("%d", z);
}
```

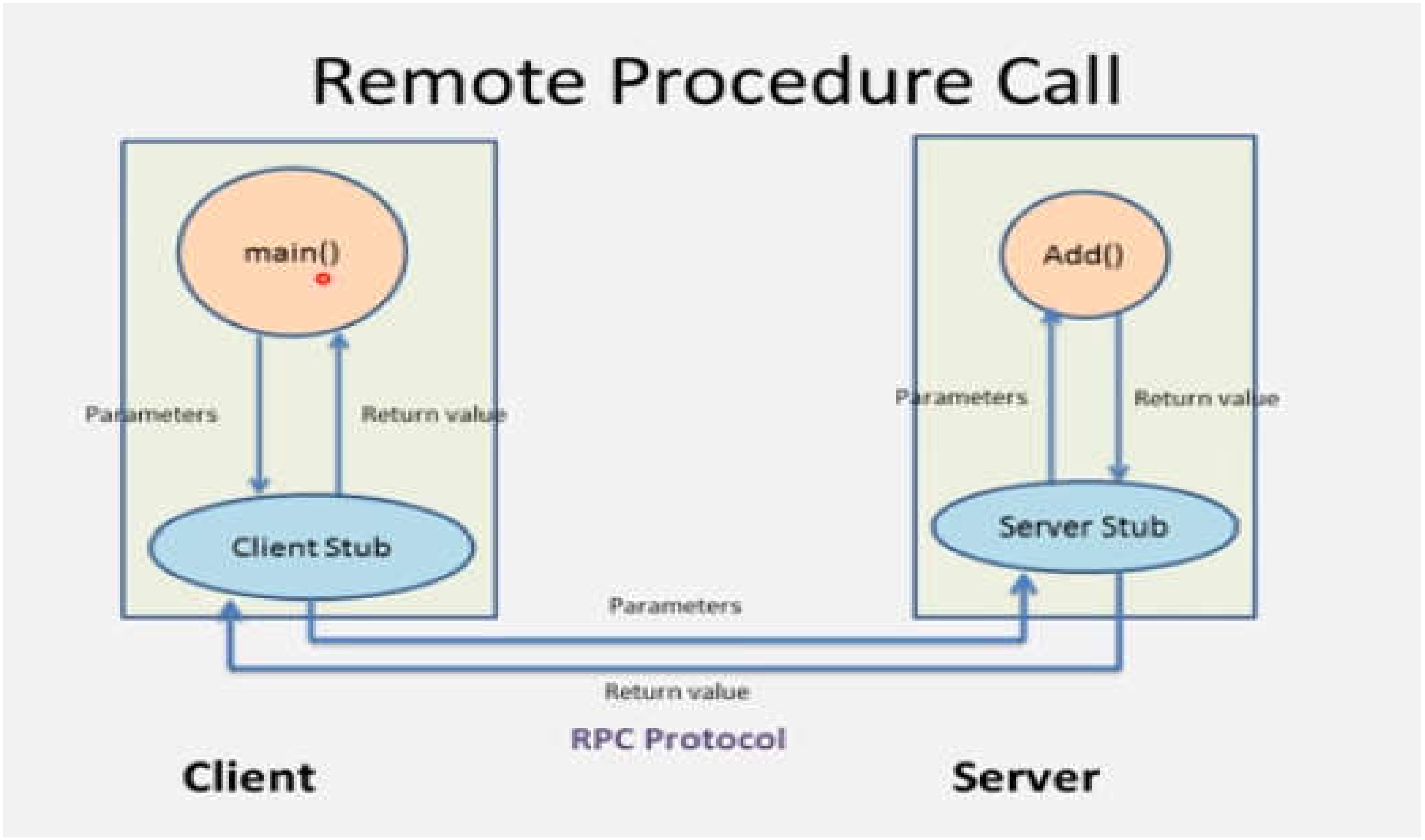
Computer 2

```
int  Add(int a, int b)
{
      int c;

      c=a+b;

      return c;
}
```

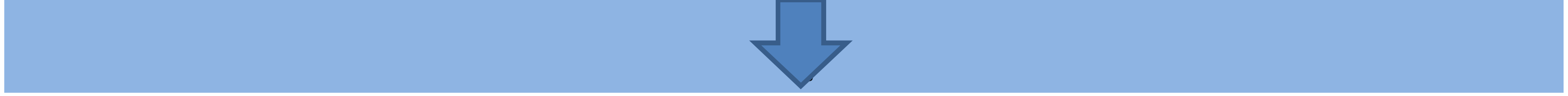
In above fig: two computers are used. One computer has main method and other contains the specific function.

In such process the task is very tedious because calling function in another computer requires concept of networking and have to write code related to networking also.

So let's see how the task can be performed easily??

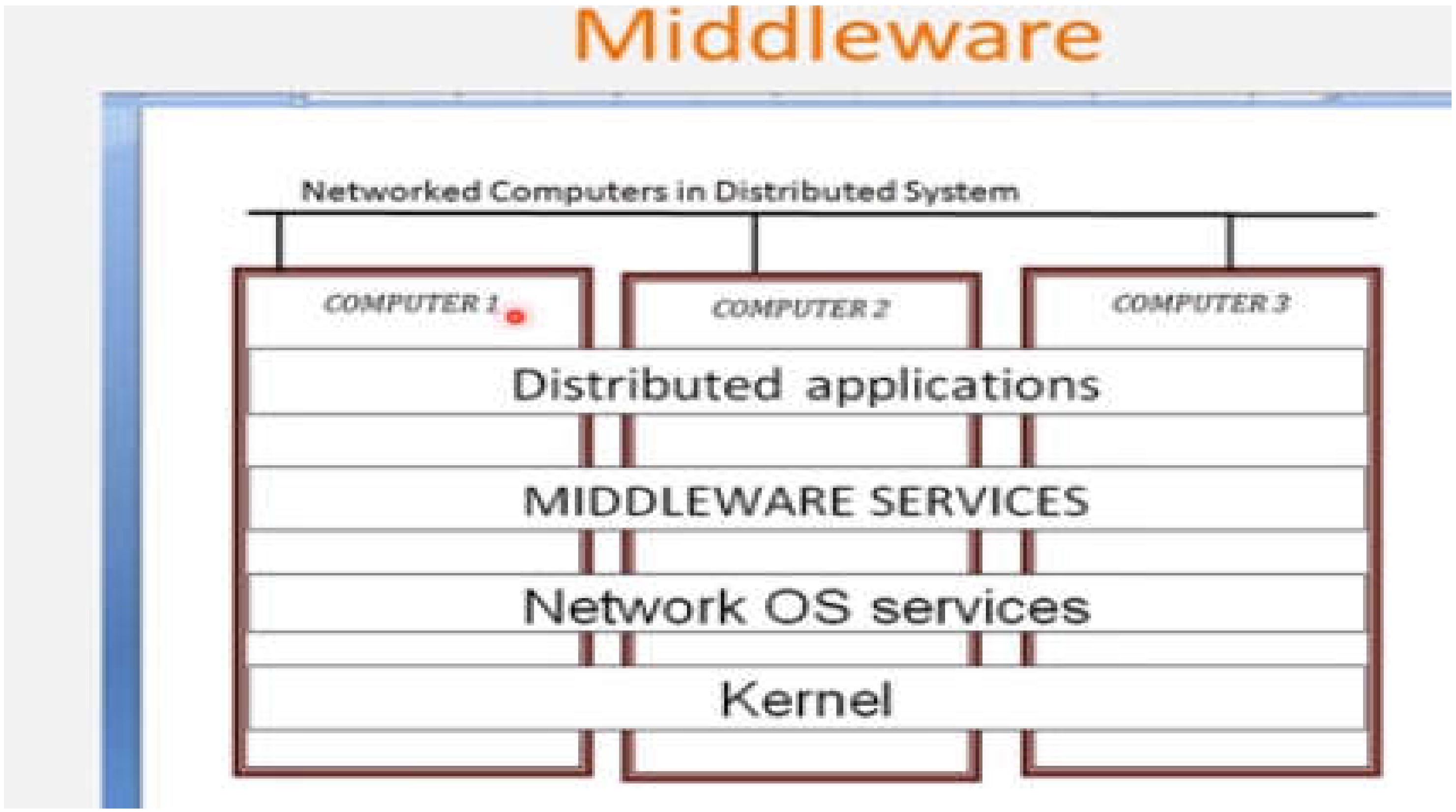


A **stub**, in the context of distributed computing, is a piece of code that is used to convert parameters during a remote procedure call (RPC). The main idea of an RPC is to allow a local computer (**client**) to remotely call procedures on a different computer (server).



- In above fig: Now main function sends parameter to client stub which then convert parameters into network portable form and process is called marshalling.
- Client stub send it to server stub and again server stub converts it to the required programming language and process is called unmarshalling.
- Same process repeat for return value as well.
- Here, main () function treats client stub as add() function and add() function treats sever stub as main() function.

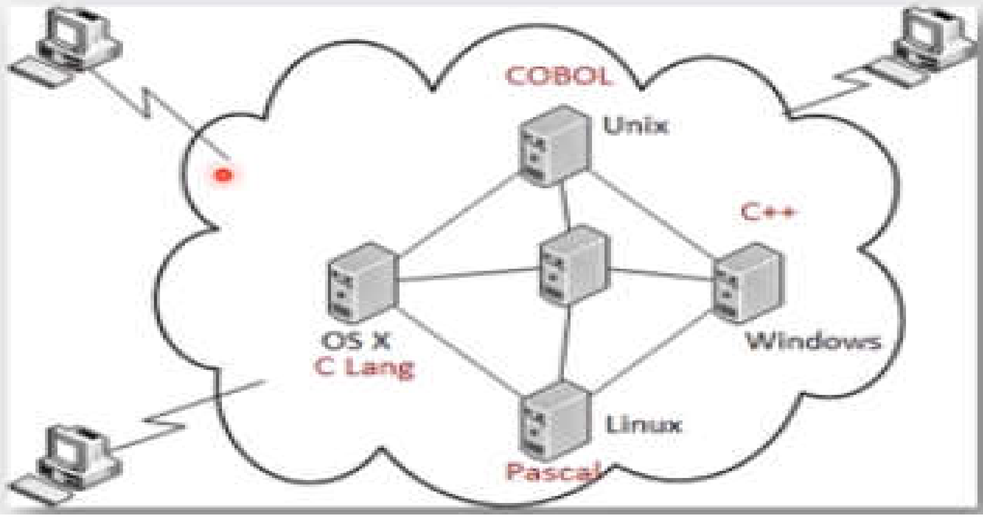
- Here, stubs and protocol acts as middleware.
- Middleware helps programmers because they donot have to write network code directly.



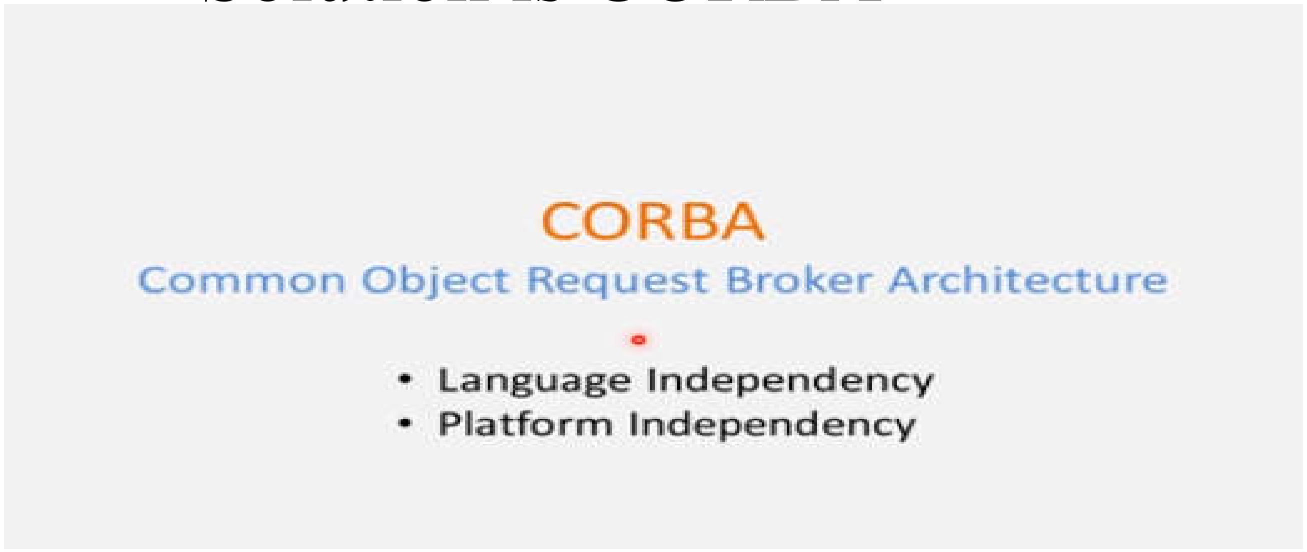
But if we need to call certain java function of one computer by another function written in C++ of second computer. How to communicate ?

Problem with Distributed Systems

- Different Operating System
- Different Languages



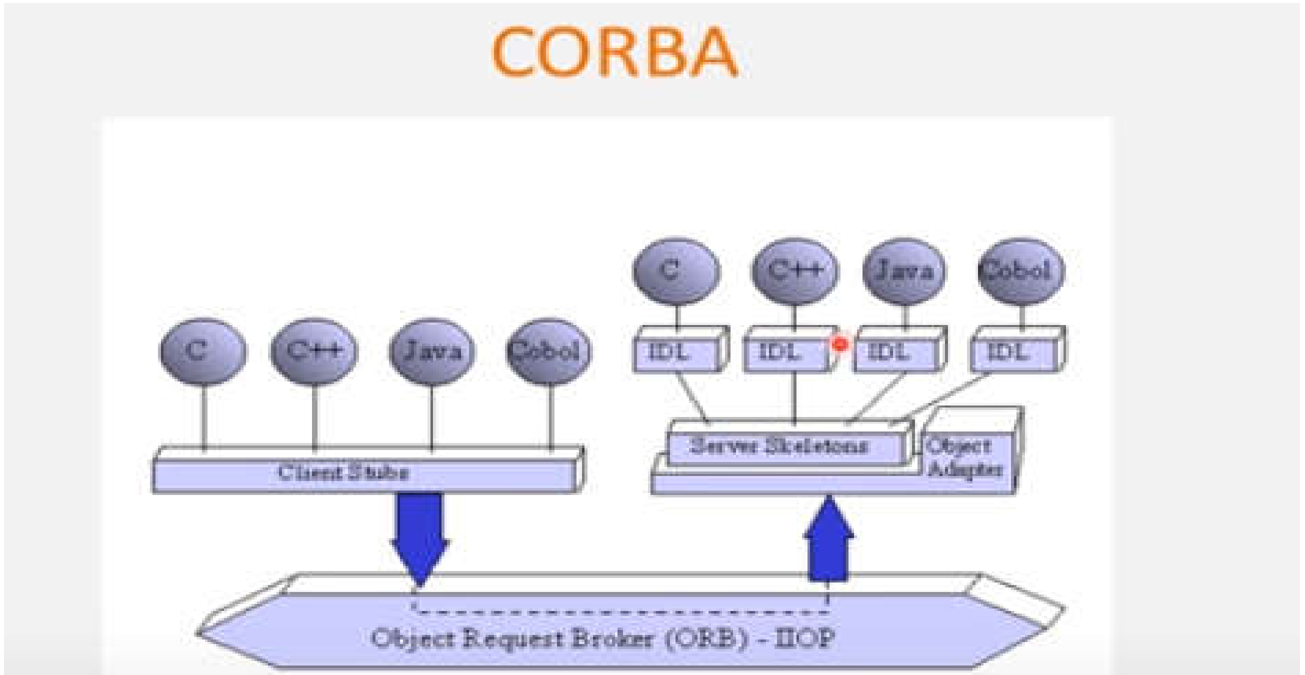
Solution is CORBA



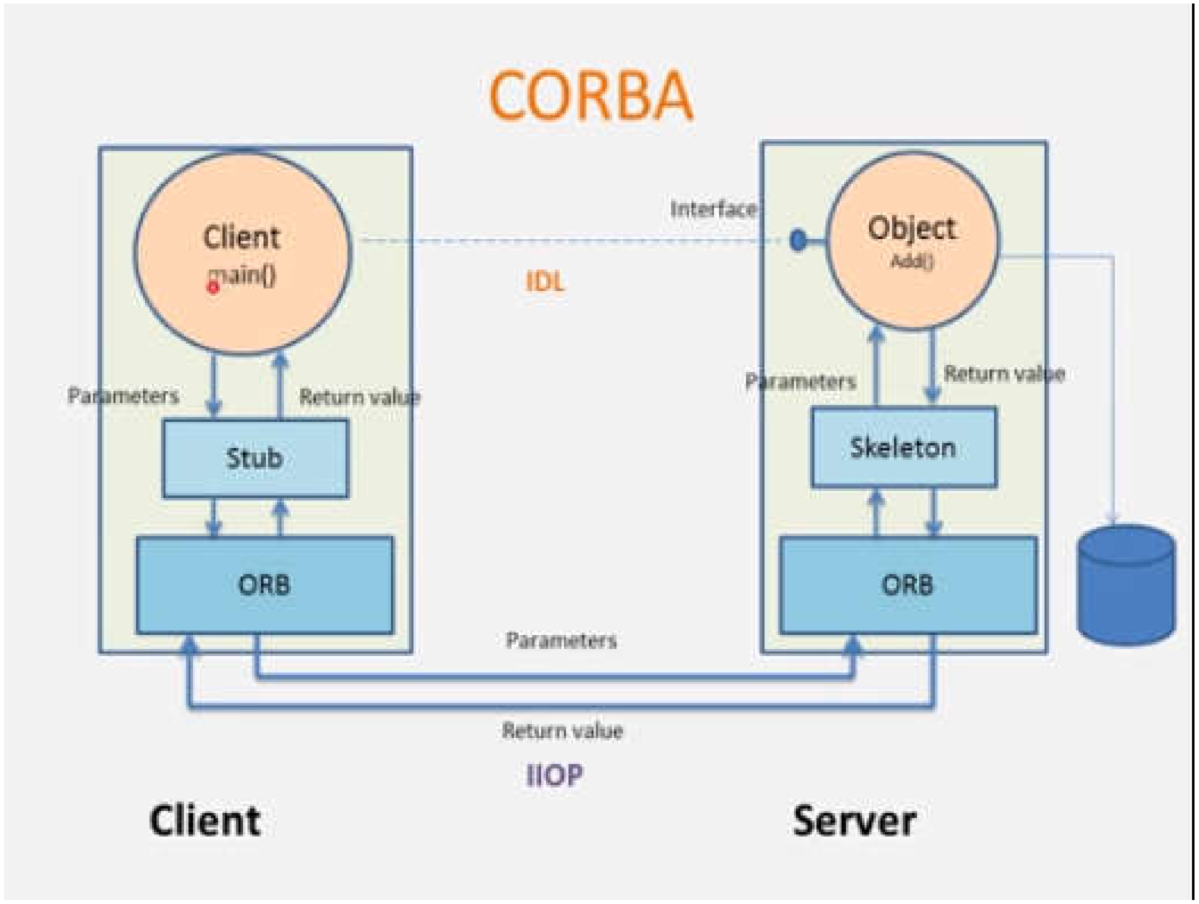
CORBA

Common Object Request Broker Architecture

- Language Independency
- Platform Independency



CORBA



- Stub = same as client stub ,
- Skeleton = same as server stub,
- ORB= platform independent so communication takes place between them where an ORB actually provides a framework which enables remote objects to be used over the network
- IDL(interface definition language)= stub converts parameters into IDL so communication takes with IDL

Similarly,

- Components express themselves through interfaces. An interface is the connection to the user that will interact with a component.
- If an interface is changed the user needs to know that it has changed and how to use the new version of it.
- Functions that are exposed to the user are usually called Application Programmable Interface (API). If there is a change to the API, the user has to recompile his code as well.

Component Model

- A component model is a definition of standards for component implementation, documentation and deployment.
- Examples of component models
  - EJB model (Enterprise Java Beans)
  - COM+ model (.NET model)
  - Corba Component Model
- The component model specifies how interfaces should be defined and the elements that should be included in an interface definition.

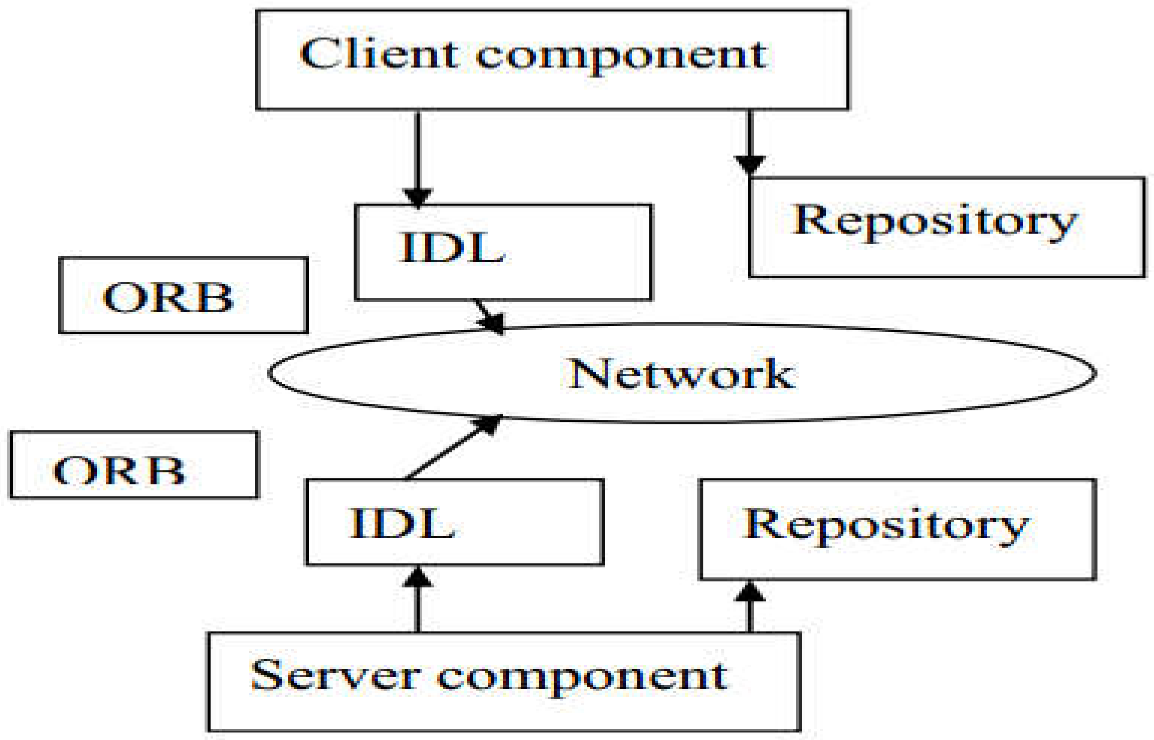
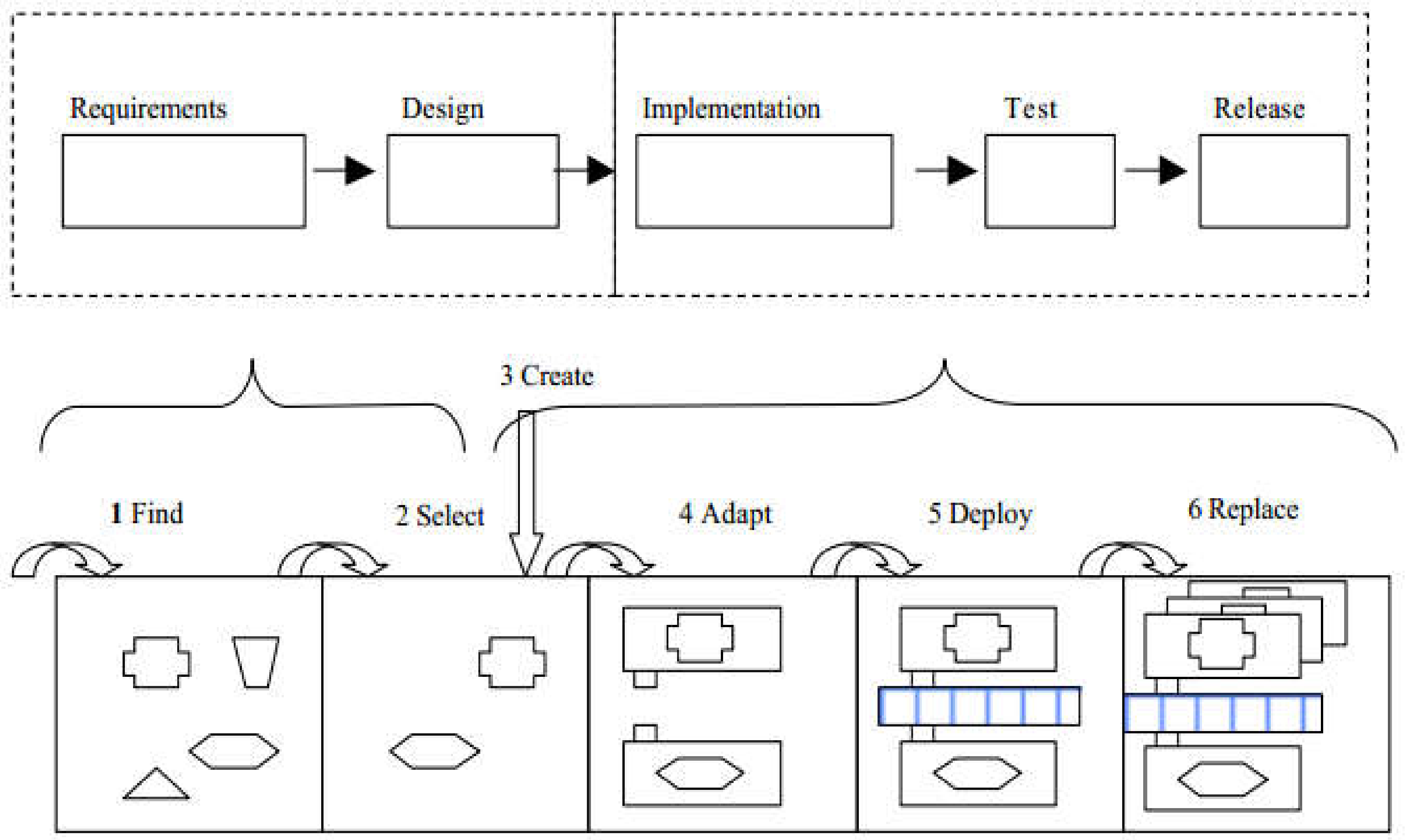


Fig: Corba Component Model

Component Based Development Process



Steps…

- The different steps in the component development process are:

  1. Finding components that may be used in the product. Here all possible components are listed for further investigation.
  2. Select the components that fit the requirements of the product.
  3. Create a proprietary component that will be used in the product. We do not have to find these types of components since we develop them ourselves.

Steps…

4. Adapt the selected components so that they suit the existing component model or requirement specification. Some component needs more wrapping than others.

5. Compose or deploy the product. This is done with a framework or infrastructure for components.

6. Replace old versions of the product with new ones. This is also called maintaining the product. There might be bugs that have been fixed or new functionality added.

.

Use of component-based development brings many advantages:

- faster development,
- lower costs of the development,
- better usability,
- to reduce the time to market,
- to meet rapidly emerging consumer demands. etc

Disadvantages:

- when you buy a component you do not know exactly its behavior,
- you do not have control over its maintenance,
- the implementation is quite hard,
- Process of improving reuse has been long and laborious etc.
- Security is another major concern for the developers who reuse the components available over the Internet. There may be a virus inside that component and may pass all the information of the business organization to attacker.

SOFTWARE REUSE

Software reuse

- In most engineering disciplines, systems are designed by composing existing components that have been used in other systems.

- Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need to adopt a design process that is based on *systematic software reuse.*

Reuse-based software engineering

- Application system reuse
  - The whole of an application system may be reused either
    - by incorporating it without change into other systems (COTS reuse) or
    - by developing application families that have common architecture
- Component reuse
  - Components of an application from sub-systems to single objects may be reused
- Object and function reuse
  - Software components that implement a single well-defined object or function may be reused

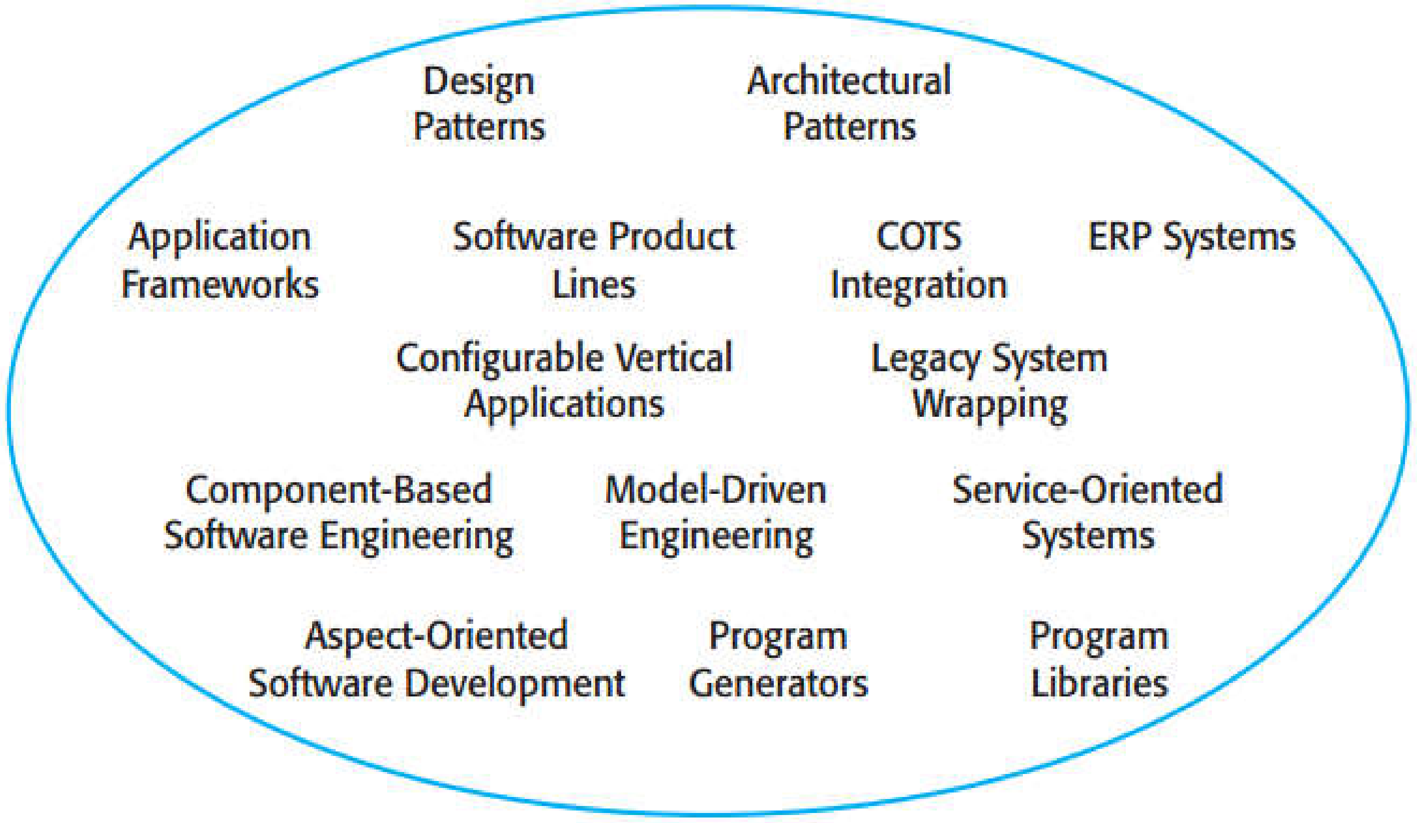
Benefits of Software Reuse

Benefit	Explanation
Increased dependability	Reused software, which has been tried and tested in working systems, should be more dependable than new software. Its design and implementation faults should have been found and fixed.
Reduced process risk	The cost of existing software is already known, whereas the costs of development are always a matter of judgment. This is an important factor for project management because it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as subsystems are reused.
Effective use of specialists	Instead of doing the same work over and over again, application specialists can develop reusable software that encapsulates their knowledge.
Standards compliance	Some standards, such as user interface standards, can be implemented as a set of reusable components. For example, if menus in a user interface are implemented using reusable components, all applications present the same menu formats to users. The use of standard user interfaces improves dependability because users make fewer mistakes when presented with a familiar interface.
Accelerated development	Bringing a system to market as early as possible is often more important than overall development costs. Reusing software can speed up system production because both development and validation time may be reduced.

Problems of Software Reuse

Problem	Explanation
Increased maintenance costs	If the source code of a reused software system or component is not available, then maintenance costs may be higher because the reused elements of the system may become increasingly incompatible with system changes.
Lack of tool support	Some software tools do not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system. The software process assumed by these tools may not take reuse into account. This is particularly true for tools that support embedded systems engineering, less so for object-oriented development tools.
Not-invented-here syndrome	Some software engineers prefer to rewrite components because they believe they can improve on them. This is partly to do with trust and partly to do with the fact that writing original software is seen as more challenging than reusing other people's software.
Creating, maintaining, and using a component library	Populating a reusable component library and ensuring the software developers can use this library can be expensive. Development processes have to be adapted to ensure that the library is used.
Finding, understanding, and adapting reusable components	Software components have to be discovered in a library, understood and, sometimes, adapted to work in a new environment. Engineers must be reasonably confident of finding a component in the library before they include a component search as part of their normal development process.

Reuse landscape



Reuse planning factors

- The development of schedule for the software
- The expected software lifetime
- The background, skills and experience of the development team
- The criticality of the software and its non-functional requirements
- The application domain
- The execution platform for the software

Design Patterns

- In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software design.
- A design pattern isn't a finished design that can be transformed directly into code.
- It is a description or template for how to solve a problem that can be used in many different situations.

Pattern elements

There are four essential elements of the design patterns:

- Name
  - A name that is a meaningful reference to the pattern
- Problem description.
  - Description of the problem & explains what patterns may be applied.
- Solution description.
  - Not a concrete design but a template for a design solution that can be instantiated in different ways.
- Consequences
  - The results and trade-offs of applying the pattern
  - Helps the designer to understand whether a pattern can be effectively applied in particular situation

Framework

- A framework is something that gives programmers most of the basic building blocks they need to make an app.
- Imagine you're cooking feast for 20 people. You're going to need an oven, a stove, a fridge, a sink, probably hundreds of ingredients, utensils, plates – etc etc.
- A framework is like a fully stocked kitchen. It has all of these things ready for you to cook with. You – or the people paying you for the meal – just need to work out what to make with it all!
- There are a few downsides to having a ready made kitchen. Maybe the oven isn't *quite* the right size, or there aren't quite enough plates, or you're lacking some ingredients, but for the most part, everything you want is in there where you can find it and you can make it work.

Framework

- Programming without a framework is like trying to build the perfect kitchen from scratch before preparing the meal.
- First you need to decide what you're going to make. If it needs an oven, you can decide to either buy the perfect oven, or build your own makeshift one.
- If your ingredients need refrigeration, you can work out some way to keep them cold.
- Maybe you like certain brands of ingredients? Well, you've got the freedom to buy just those brands, instead of what a pre-stocked kitchen might give you work.

- A software framework is an all inclusive, reusable programming environment that gives specific usefulness as a major aspect of a bigger programming stage to encourage advancement of programming applications, items and arrangements.

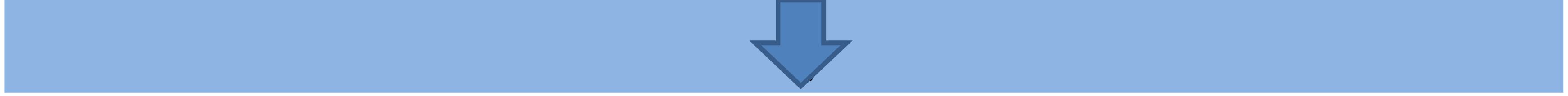
- Software frameworks may incorporate bolster programs, compilers, code libraries, device sets, and application programming interfaces (APIs) that unite all the diverse segments to empower advancement of an undertaking or arrangement.

Simple Example of MVC Pattern In Real Life

- Imagine a photographer with his camera in a studio. A customer asks him to take a photo of a box.
- The box is the *model*, the photographer is the *controller* and the camera is the *view*.
- Because the box does not *know* about the camera or the photographer, it is completely independent. This separation allows the photographer to walk around the box and point the camera at any angle to get the shot/view that he wants.

MVC PATTERN

- The model manages fundamental behaviors and data of the application. It can respond to requests for information, respond to instructions to change the state of its information, and even to notify observers in event-driven systems when information changes. This could be a database, or any number of data structures or storage systems. In short, it is the data and data-management of the application.

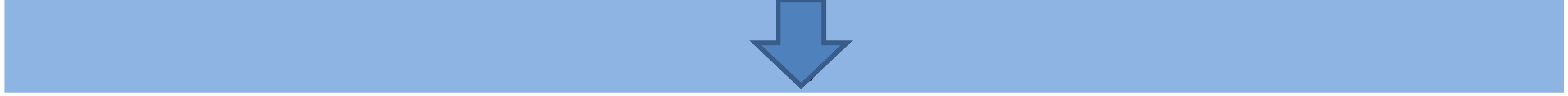


- The view effectively provides the user interface element of the application. It'll render data from the model into a form that is suitable for the user interface.
- The controller receives user input and makes calls to model objects and the view to perform appropriate actions.
- All in all, these three components work together to create the three basic components of MVC.

Now let's say we have an online Banking system, from where the user needs to check his account balance.

View: The UI form which the end user sees and sends the request from. Typically in this case it could either be the online web browsers or the mobile UI, from where the end user sends the request to check his balance.

Controller: Now what if the user desires to do an online fund transfer from one account to another. In this case you would be needing a whole lot of business logic, that accepts the user request, checks his balance in Account 1, deducts the funds, transfers to Account 2, and updates the balance in both cases. What the Controller part here essentially does is accept the request from user to transfer funds, and re direct it to the necessary components that would do the job of transfer.



Model:

Here it responds to requests from users to just read the data(handled from the view) or do an update of the data(handled by the controller). In this case the Model, would be the part of the application that interacts with the database here either to read or write the data. So the user makes a request from the browser to check his balance amount, the Model would be the part of the application, that receives it either from view, processes the request, and sends the data back.

Chapter Five and Six finished !!!

Nearly 5 to 10 marks

To be continued……………..

!! Attendance please !!

Thank You!!!