Chapter 4: Software Design

System Design: Introduction

- **Introduction:**

- **Systems design** is the process of **describing, organizing, and structuring** the components of a system at both **the architectural level and a detailed level**, with a view toward constructing the proposed system.

- **Systems design is like a set of blueprints used to build a house**. The blue prints are organized by the different components of the house, and describe the rooms, walls, windows, doors, wiring, plumbing, and all other details.

- Design indicates that it involves **describing, organizing, and structuring** the system solution.

- **The output of the design** activities **is a set of diagrams and documents** that achieves this objective. These diagrams model and document various aspects of the solution system.

- **Elements of Design**

  To understand the various elements of systems design, we must consider two questions:

  1. What are the **components** that require systems design?
  2. What are the **inputs to and outputs** of the design process?

  - **Components of System Design**

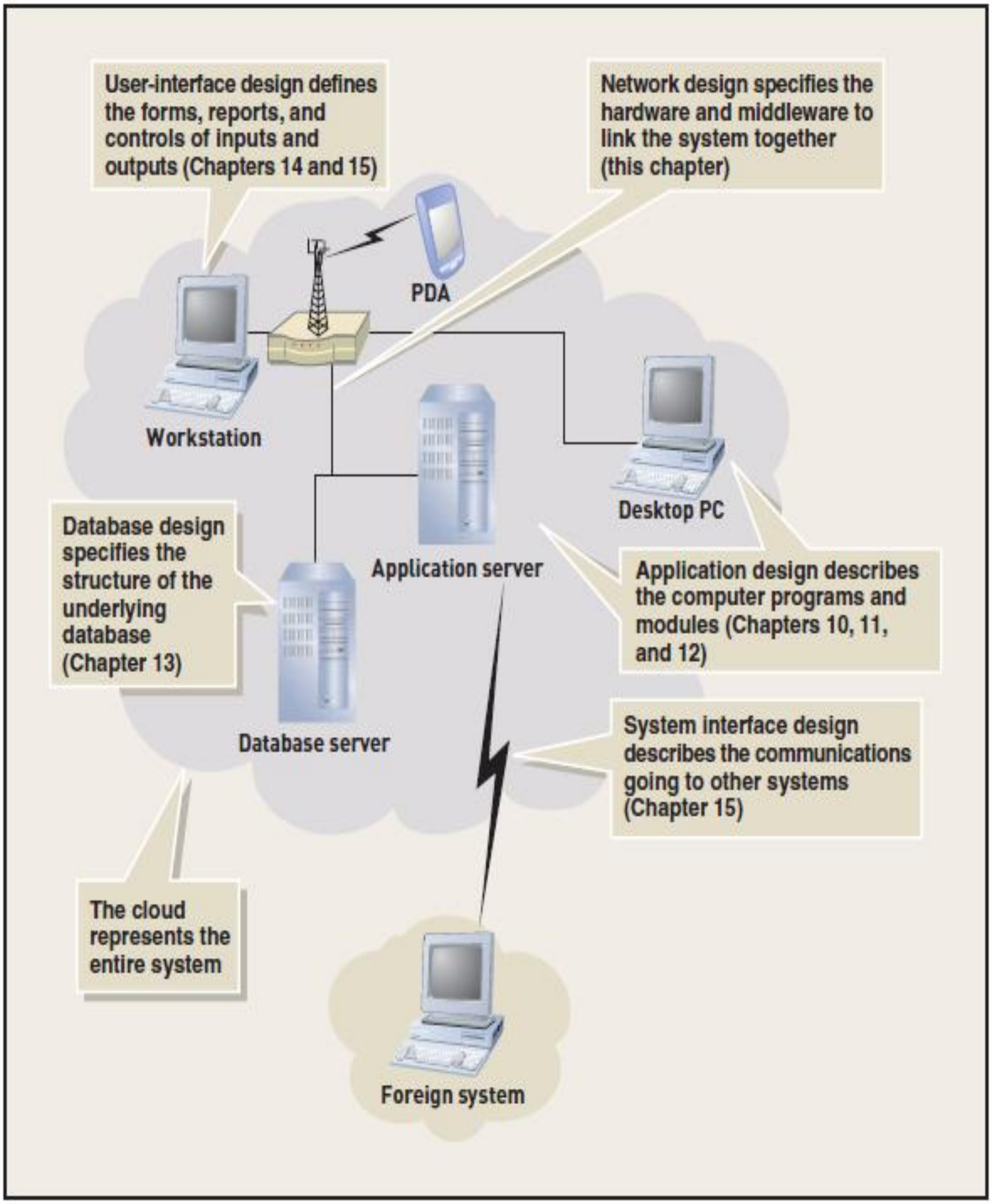


Figure: Components of System Design

- **Elements of System Design**
    - **Design is much more oriented toward technical issues and therefore requires less user involvement and more involvement by other systems professionals.**

System Development Approach	Inputs
1. Structured Approach	• data flow diagrams • entity-relationship diagrams
2. Object Oriented Approach	• Use Case Diagram • Class Diagram

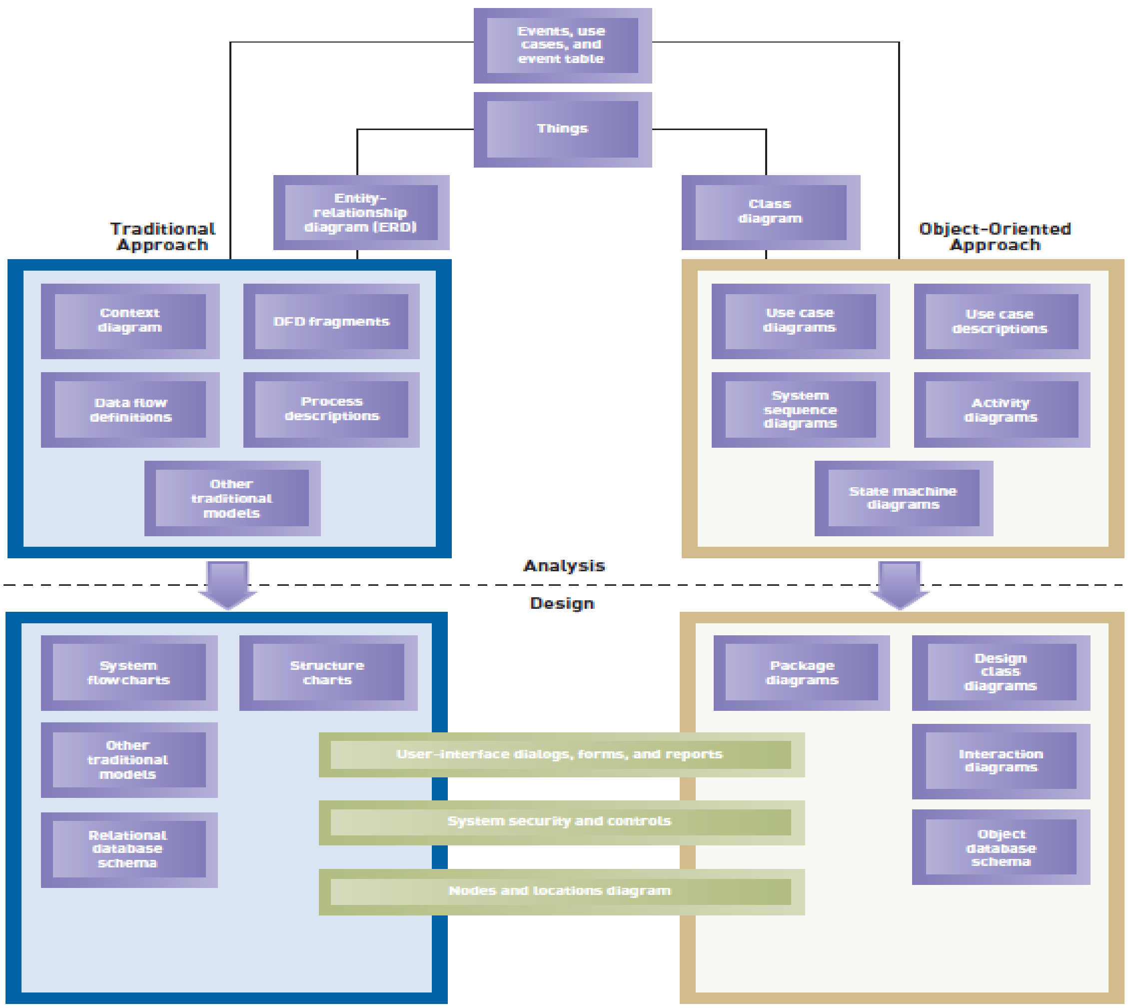


Figure: Input of the system Design

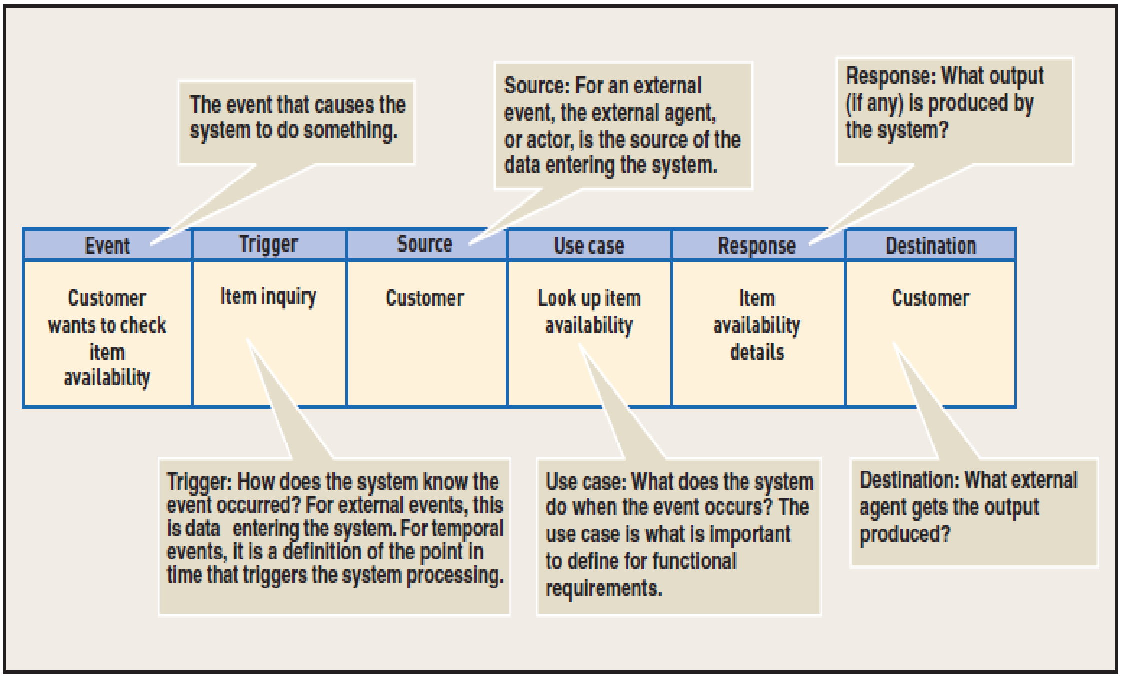


Figure: Event Table

- ## Design Activities

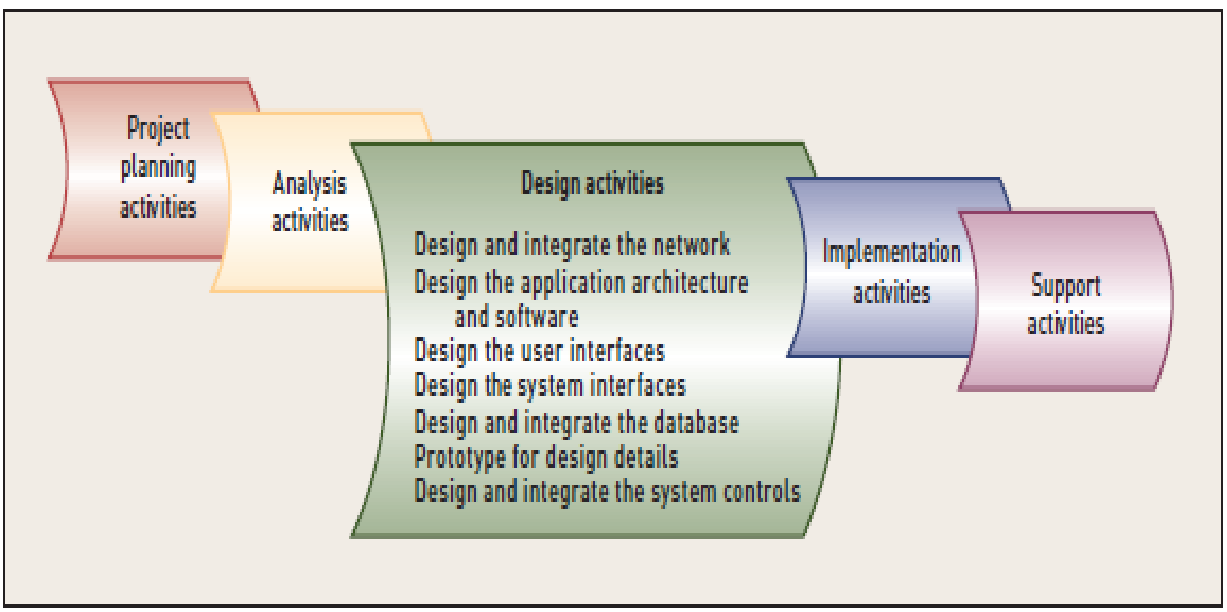


Figure: SDLC with various Design activities

- ## Software Design:

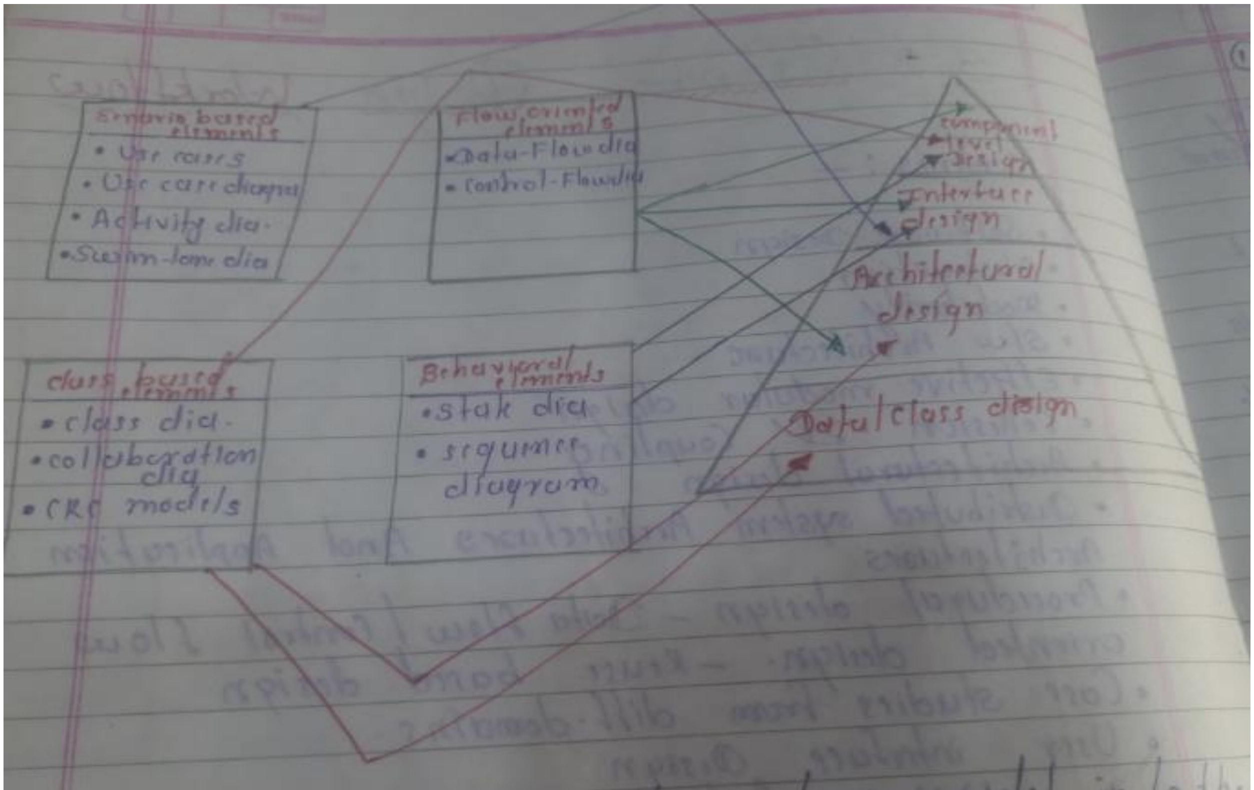


fig:- Translating the analysis model into the design model

Design → • The process of applying various techniques & principles for the purpose of defining a device, a process or a system in sufficient detail to permit it's physical realization.

• Design is an iterative process through which requirements are translated into a "blueprint" for constructing the s/w.

Types of Software Design:

(1) Data Design → • Transforms the information domain model created during analysis in to the data structures that will be required to implement the software.

• The data objects & relationships defined in the entity-relationship diagram & the detailed data content provided in data dictionary provide the basis for the data design activity.

(2) Architectural Design →

• Architectural design defines the relationship among major structural elements of the program.

(3) The interface Design →

• Interface design describes how the s/w communicates within itself, to system that interoperate with it, & humans who use it.

with

(4) The Procedural Design →

• Procedural design transforms structural elements of the program Architecture in to a procedural description of s/w components.

(5) Presentation Design →

• media designers define the look and to some extent – the structure of how multimedia contents are presented.

Characteristics of Good Design:

Characteristics of Good design

Throughout the design process, the quality of the design is assessed with a series of formal technical reviews or design walkthroughs.

1> The design must implement all of the explicit requirements contained in the analysis model, & it must accommodate all of the implicit requirements desired by the customer.

2> The design must be readable, understandable guide for those who generate code & for those who test & maintain the s/w.

3> The design should provide a complete picture of the s/w, addressing the data, functional & behavioral domains from an implementation perspective.

Principles of Good Design

Principles of Good Design

1> Divide & Conquer → Dividing the project or system into seperate modules & seperate people can work on each part. Original development can therefore done in parallel.

2> Increase cohesion where possible →

3> Reduce coupling →

4> keep level of abstraction as high as possible →

5> Increase reusability where possible. →

6> Design for flexibility →

7> Anticipate obsolescence → means planning for changes in the technology or envirornment, so s/w will continue to run or can be easily changed.

8> Reuse existing designs & code where possible

9> Design for portability →

10> Design for testability →

11> Design Defensively → check all i/p's to your system components are valid or not.

Design Concepts :

1> Abstraction → (presentation of simple concept or object to the external world)

- The process of hiding unessential information & delivering essential information.
- When we consider a modular solution to any problem, many levels of abstraction can be possible.
- At the highest level of abstraction, a solution is stated in broad terms using the language of problem environment.
- At the lowest level of abstraction, more detailed description of the solution is provided.
- There are diff. types of abstraction :

i> procedural abstraction →
Refers to sequence of instructions that have a specific & limited functions; but specific details are supressed.
[Procedure abstraction is to decompose problem to many simple subunits]

eg> Consider an example of word open for a door. Open implies a long sequence of procedural steps. i.e.
a> walk to the door
b> turn knob & pull door
c> step away from moving door.

ii> Data Abstraction →
Is a named collection of data that describe a data objects.
e.g.> Data abstraction for door consist of a set of attributes that describe the door. i.e.
a> door type b> swing direction c> opening mechanism d> weight e> dimensions

2) Software Architecture →

- s/w architecture is the s/w engineering activity concerned with designing & specifying the overall system structure.
- s/w architecture of a application or a computing system is the structure of the system, which comprise of s/w elements, externally visible qualities of those elements & relationships among them.

▸ Why s/w Architecture is important?

i) s/w architecture helps us to communicate the system design to project stakeholders. (users, managers, implementers)

ii) It helps us to analyze design decisions.

iii) It helps us reuse concepts & components in future system.

iv) Exposes the structure of the system but hide implementation details.

▸ Role of s/w Architect

i) Central technical communicator

ii) key decision maker

iii) Responsible for delivery

fig:- s/w architect as communicator

Difference between Abstraction & Encapsulation →

① Data abstraction is a process through which we can remove unnecessary details · whereas data encapsulation is wrapping up of data into a single frame called class; so that it can be reused.

② Data abstraction allows programmers to think simply about a problem.

③ Abstraction enables a designer to specify procedure & data & yet suppress low-level details.

3> modularity →

- s/w is divided into separately named & addressable components, called modules
- Dividing s/w system into pieces; separate people can work on each part; so original development work can done in parallel.
- One becomes expert in particular module, it is not possible for someone to know everything.
- Each component is smaller & therefore easier to understand.
- When one part needs to be replaced or changed, it can hopefully be done without having to replace or extensively change other parts.

4) Refinement →

- In each step of refinement, one or several instructions of the given program are decomposed into more detailed instructions.
- Refinement is actually a process of elaboration.
- Refinement causes the designer to elaborate on the original statement, providing more & more detail as each successive refinement occurs.
- Abstraction & refinement are complementary concepts. Abstraction enables a designer to specify procedure & data & yet suppress low-level details. Refinement helps designer to elaborate low-level details as design progresses.

5) Information hiding →

- Information contained within a module is inaccessible to other modules; that have no need for such information.

6) Effective modular Design →

- Design s/w in such a way that; each module addresses specific subfunction of requirements & has simple interface when viewed from other parts of the program structure.
- S/w with effective modularity i.e. independent modules is easier to develop. Independent modules are easier to maintain & test; because secondary effects caused by design /code modification are limited, error propagation is reduced reusable modules are possible. & this is called as functional independence & it results in good design & good design is key to software quality.
- Functional independence is measured using 2 criteria:

i> Cohesion → is a ~~[illegible]~~ measure of relative functional strength of a module.

ii> Coupling → is a measure of relative interdependence among modules.

i> Cohesion →

1. functional cohesion
2. Logic cohesion
3. communicational cohesion
4. sequential cohesion
5. procedural cohesion
6. Temporal cohesion
7. utility cohesion

ii> Coupling →

1. Content coupling
2. common coupling
3. control coupling
4. stamp coupling
5. Data coupling

Cohesion → is the measure of strength of associati[on of] elements within a module.

→ module is highly cohesive if its elem[ents] strongly related to each other.

Modularization

Modularization is a technique to **divide a software system into multiple discrete and independent modules,** which are expected to be capable of carrying out task(s) independently.

These modules may work as basic constructs for the entire software. **Designers tend to design modules such that they can be executed and/or compiled separately and independently.**

Modular design unintentionally follows the rules of 'divide and conquer' problem-solving strategy this is because there are many other benefits attached with the modular design of software.

Advantage of modularization:

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again.
- Concurrent execution can be made possible

Coupling and Cohesion

When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together. There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

- **Co-incidental cohesion -** It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.
- **Logical cohesion -** When logically categorized elements are put together into a module, it is called logical cohesion.
- **Temporal Cohesion -** When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
- **Procedural cohesion -** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
- **Communicational cohesion -** When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- **Sequential cohesion -** When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
- **Functional cohesion -** It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Coupling

Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

There are five levels of coupling, namely -

- **Content coupling -** When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
- **Common coupling-** When multiple modules have read and write access to some global data, it is called common or global coupling.
- **Control coupling-** Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.
- **Stamp coupling-** When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
- **Data coupling-** Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

Ideally, no coupling is considered to be the best.

1) Coincidental cohesion → no apparent relationship among module elements
(Weakest cohension)

2) Logical cohesion → some inter-elements relationships exist such as several related functions, math library,
(several logically related elements)

3) Temporal cohesion → elements are usually bounded through logic and are executed at one time.

4) procedural i.e. same invocation of module

4) Communication cohesion → all elements are executed at one time and also refer to the same data, such as I/O module.
i.e. all elements access same data that are defined within one class.
* elements contribute to activities that use same input or o/p data

5) Sequential cohesion → o/p of one element is i/p to next
i.e. module structure bear close resemblance.

6) function cohesion → all elements relate to performance of a single function.
all elements contribute to execution of one and only one problem-related

7) information cohesion → complex data structure with all it's functions / operators

highest → to lowest cohesion

* Functional Cohesion → i) All elements contribute to the execution of one and only one problem related task

(2) No elements doing unrelated activities

example → read transaction record, assign seat to airline passenger

(2) Sequential Cohesion → o/p of one element become i/p of other elements and so on.

example → module format and cross-validate record
→ use raw record, format raw record, cross validate field in raw record
→ end of record

(3) communicational → elements contribute to activities that use same i/p and o/p data.

example → module determine customer details
• use customer account no • find customer name
• find customer loan balance • return name and loan balance
→ End module

(4) procedural Cohesion → Elements are related only by sequence, otherwise activities are unrelated

example → module read, write and edit something
→ use record → write out record → read record
→ pad numeric field with zeros
→ return record
→ end module

5) Temporal Cohesion → Elements are performed some activities that are related in time
→ Temporal cohesion commonly found in initialisation and termination modules → elements are basically unrelated and hence difficult to reuse.

Example → module initialise
- set counter to 0
- open student file
- clear error message
- initialise array
- end module

6) Logical cohesion → Example ✓ module display record
- use record type
- if record-type = student then
  display record student
  else if record-type = staff then
  display staff-record
  end module

Architecture Design

- The second activity identified in design phase of SDLC is to design the application architecture and software.
- There are various types of Application Architecture.

1. **Single Computer Architecture**
2. **Multitier Architecture**
3. **Centralized and Distributed Architecture**
4. **CLIENT/SERVER ARCHITECTURE**
5. **THREE-LAYER CLIENT/SERVER ARCHITECTURE**
6. **INTERNET AND WEB-BASED APPLICATION ARCHITECTURE**
7. **WEB SERVICES ARCHITECTURE**
8. **Middleware**

1. **Single Computer Architecture:**
   - **Single-computer architecture** employs a single computer system and its directly attached peripheral devices, as shown in Figure. Even though single-computer architecture can refer to a stand-alone PC, a single PC has limited capabilities even at today's computer speeds.
   - **This Architecture Employs Single computer system executing all application related software**
   - The primary advantage of single-computer architecture is its simplicity.
   - Information systems deployed on a single-computer system, even though the software maybe complex, usually do not have complex interactions with other systems and thus operate in a less complex and less cluttered environment.
   - The other major advantage of a mainframe architecture is the extremely high-volume processing that is supported.

2. **Multitier architecture:**
   - **This Architecture distributes all application related software or processing load across multiple computer system.**
   - Employs multiple computer systems in a cooperative effort to meet information-processing needs.
   - Multitier architecture can be further subdivided into two types:

     - **Clustered architecture**

✓ **Group of computers of same type that share processing load and act as a single large computer system.**

  - ✓ Employs a group (or cluster) of computers, usually from the same manufacturer and model family. Programs are allocated to the least utilized computer when they execute so that the processing load can be balanced across all machines.
  - ✓ In effect, a cluster acts as a single large computer system. Clustered computer systems are normally located near one another so that they can be connected with short high-capacity communication links.

- **Multicomputer architecture**,
  - ✓ Employs multiple computer systems. But hardware and operating systems are not required to be as similar as in a clustered architecture.

- ✓ **Group of dissimilar computers that share processing load through specialization of function**
- ✓ **A suite of application or system programs and data resources is exclusively assigned to each computer system**.
- ✓ Each computer system is optimized to the role that it will play in the combined system, such as database or application server.

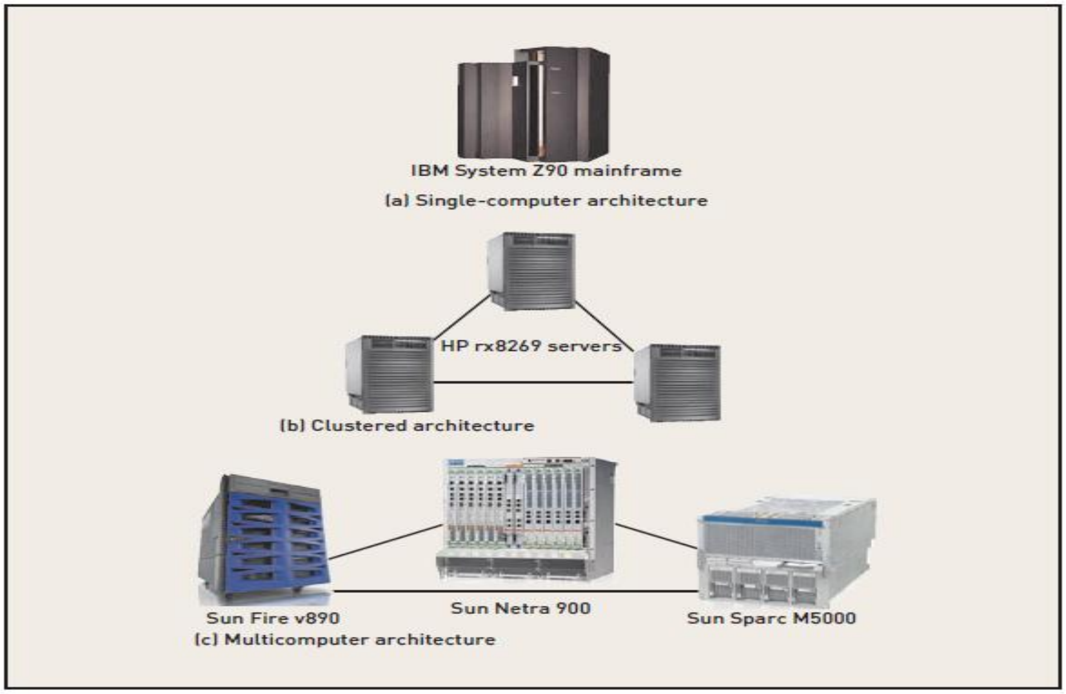


Figure: Single Computer, Clustered and Multicomputer Architecture

3. CENTRALIZED AND DISTRIBUTED ARCHITECTURE:

- ➢ The term **centralized architecture** describes deployment of all computer systems in a single location.
- ➢ **Architecture that locates all computing resources in a central location.**
- ➢ Centralized architecture is generally used for large-scale processing applications, including both batch and real-time applications.
- ➢ Such applications are common in industries such as banking, insurance, and catalog sales.

4. CLIENT/SERVER ARCHITECTURE

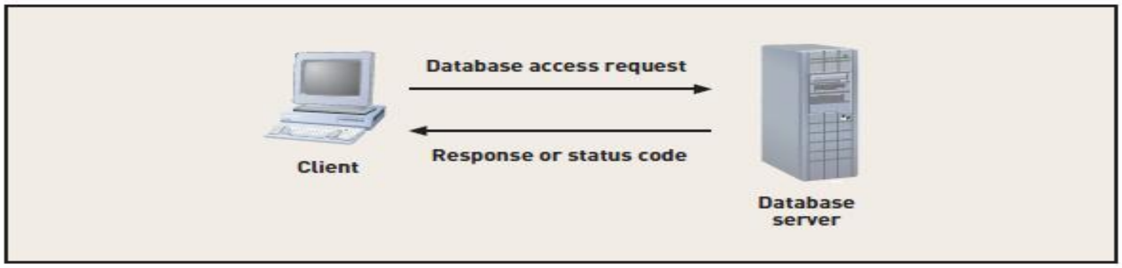


Figure: Client Server Architecture

- **Client/server architecture divides programs into two types: client and server.**
- **A server manages one or more information system resources or provides a well-defined service.**
- **A client communicates with a server to request resources or services, and the server responds to those requests.**
- Client/server architecture is a general model of software organization and behavior that can be implemented in many different ways.
- A typical example is the interaction between a client application program executing on a workstation and a database management system (DBMS) executing on a larger computer system. The application programs end database access requests to the database management system via a network. The DBMS accesses data on behalf of the application and returns a response such as the results of a search operation or the success or failure result of an update operation.

- When designing client/server software, the following architectural issues must be addressed:
    - ✓ Decomposing the application into client and server programs, modules, or objects
    - ✓ Determining which clients and servers will execute on which computer systems
    - ✓ Describing the communication protocols and networks that connect clients and servers.

5. THREE-LAYER CLIENT/SERVER ARCHITECTURE

- A widely applied variant of client/server architecture, called **three-layer architecture**, **divides application software into a set of client and server processes independent of hardware or locations.**

- All layers might reside on one processor, or three or more layers might be distributed across many processors. In other words, the layers might reside on one or more tiers.
- The most common set of layers includes the following:
  - ✓ The **data layer**, which manages stored data, usually in one or more databases
  - ✓ The **business logic layer,** which implements the rules and procedures of business processing
    - ✓ The **view layer,** which accepts user input and formats and displays processing results.
- Three-layer architecture is currently a widely applied architectural design pattern with both the traditional approach and the object-oriented approach.

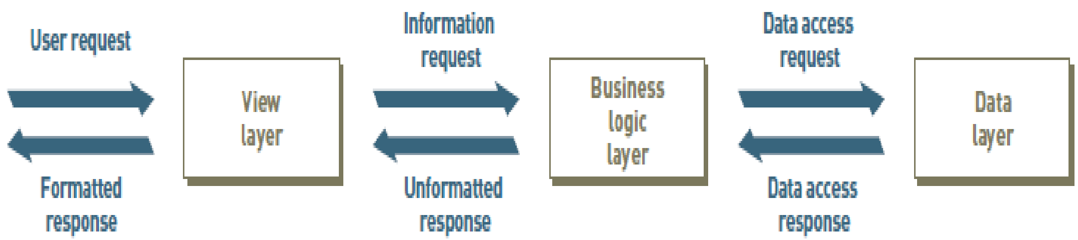


Figure: Three layer Client Server Architecture

6. INTERNET AND WEB-BASED APPLICATION ARCHITECTURE

- The Web is a complex example of client/server architecture. Web resources are managed by server processes that can execute on dedicated server computers or on multipurpose computer systems.
- Clients are programs that send requests to servers using one or more of the standard Web resource request protocols. Web protocols define valid resource formats and a standard means of requesting resources and services. Any program, not just a Web browser, can use Web protocols. Thus, Web-like capabilities can be embedded in ordinary application programs.
- Internet and Web technologies present an attractive alternative for implementing information systems.

7. WEB SERVICES ARCHITECTURE

- Web services architecture is another modern variant of client/server architecture.
- **Web services architecture** packages software into server processes that can be accessed via Web protocols, including **XML, SOAP, Web Services Description Language (WSDL), and Universal Description, Discover, and Integration (UDDI).**
- Information about a Web service, such as server and service names and port numbers, XML data formats, and security requirements, is described in WSDL and published in a Web services directory. The client interacts with the directory to determine what services are available and how to interact with them. The client then initiates a connection with the Web service using SOAP and XML

8. MIDDLEWARE

- Client/server and three-layer architecture relies on special programs to enable communication between the various layers.
- Software that implements this communication interface is usually called **middleware**.
- Middleware connects parts of an application and enables requests and data to pass between them.
- There are various methods to implement the middleware functions. Some common types of middleware include transaction processing monitors, object request brokers (ORBs), and Web service directories.
- Each type of middleware has its own set of protocols to facilitate communication between various components of an information system.

User and System Interface Design

Traditional approach:

- Inputs and outputs are shown as **data flows** on the context diagram, the data flow diagram (DFD) fragments, and the detailed DFDs.
- A data flow definition that lists all data elements describes each input and output in detail.
- **During design, analysts add more detail about the data flows based on the choices they made when deciding on a design alternative.**

Object-oriented approach:

- Inputs and outputs are defined by messages entering or leaving the system.
- Inputs and outputs are included in the event table as triggers and responses.
- Actors provide inputs for many use cases, and many use cases provide outputs to actors.

User Interface Design

- User interface is the means by which the user and a computer system interact, in particular the use of input devices and software.
- A **user interface**, also called a "UI" or simply an "**interface**," is the means in which a person controls a software application or hardware device.
- **Interface design based on tasks as expressed in use cases**.
- Ensure that user always knows what he or she can do next and what will happen when he or she does it.
- **UI provide good feedback**, including effective error messages, if some operation is taking more than few seconds then provide progress bar, so user knows what is going on.
- **Ensure that appearance of UI is neat**.

- **Provide all necessary helps.**
- **Interface design focuses on**
  - ✓ Design of interfaces between software modules
  - ✓ Design of interfaces between software and other non human producers and consumers of information
  - ✓ Design of interface between human and computer.

Evaluating User Interface

User interface is evaluated for any u**sability defects.**

Usability Defect: Assume a UI consist of 2 fields' login name and password. Assume you have given valid user name and invalid password and clicked on Login Button, in return if application has generated an error message stating that"Please enter valid user name", then this is usability defects.

i) Heuristic Evaluation:

- Heuristic evaluation involves systematically examining the system and looking for usability defects.
- You can perform a heuristic evaluation of finished system or a paper prototype.
- Each UI is checked for usability defects.
- Several evaluators perform heuristic evaluations independently.
- If you discover defects write down a short description of that defect and include screen shots of those defects.
- Write down your ideas for how defects might be fixed or communicate with other software engineer who will be fixing the defects

ii) Heuristic Evaluation by observation of users:

User Interface Design Process:

- User interface design process is divided in to various phases.

i) **Functionality Requirement gathering:** identify functional requirements expected by the users. After analyzing the requirements a decision is made on how to incorporate these requirements in to the interface and what functionality to include i.e. requirements are mapped in to interfaces

ii) **User Analysis:** refers to analyzing potential users of the system either through discussions with people who work with the users or with potential users themselves. Ask questions regarding;
   - ✓ What the user wants?
   - ✓ How the user is going to use the system
   - ✓ What are technical abilities of the user?

iii) **Information Architecture:** identify information architecture of the system i.e. how the information flow from one page to another page, that is to show hierarchy of pages.

iv) **Design User Interface**:

v) **Usability Testing**: Perform usability testing of user interface for any usability defects

■ User Interface Versus System Interfaces

In both the traditional and object-oriented approaches, a key step in systems design is to classify the inputs and outputs for each event as either a system interface or a user interface.

System interfaces:

- **Involve inputs and outputs that require minimal human intervention.**
- They might be inputs captured automatically by special input devices such as scanners, electronic messages from another system, or batch processing transactions compiled by another system.

.

User interfaces:

- **Involve inputs and outputs that more directly involve a system user.**
- A user interface enables a user to interact with the computer to record a transaction, such as when a customer service representative records a phone order for a customer.
- Sometimes outputs are produced after user interaction, such as the information displayed after a user query about the status of an order.
- In Web-based systems, a customer can interact directly with a system to request information, place an order, or look up the status of an order.

  The user interface is everything the end user comes in contact with while using the system.

■ Guidelines for Designing User Interfaces:

- **Visibility means that a control should be visible so users know it is available**, and that the control should provide immediate feedback to indicate it is responding.

  For example, a button on form that can be clicked by a user is visible, and when it is clicked, it changes to look as though it has been pressed to indicate it is responding. Some buttons make a clicking sound to provide feedback.

- **Affordance means that the appearance of any control should suggest its functionality**—that is, the purpose for which the control is used.

  For example, on the computer, a button affords clicking, a scroll bar affords scrolling, and an item in a list affords selecting.

- **If user-interface designers make sure that all controls are visible** and clear in what they do, the interface will be usable.

- Most users are familiar with the Windows interface and the common Windows controls. However, designers should be careful to apply these principles of visibility and affordance when designing Web pages.

- Many new types of controls are now possible at Web sites, but these controls are not always as visible and their effects are not always as obvious as they are in a standard Windows interface.

- More objects are clickable, but it is not always clear what is clickable, when a control has recognized the click, and what the click will accomplish. For example, sometimes you click on an image and a new page opens in the browser. Other times you click on an image and nothing happens.

■ Eight Golden Rules for Designing Interactive User Interfaces

Shneiderman, another leading researcher in HCI suggested eight golden rules for designing user interfaces.

1) Strive for Consistency

- ✓ Designing a consistent-appearing and -functioning interface is one of the most important design goals.

- ✓ The way that information is arranged on forms, the names and arrangement of menu items, the size and shape of icons, and the sequence followed to carry out tasks should be consistent throughout the system.

2) Enable Frequent Users to Use Shortcut

- ✓ Users who work with one application all day long are willing to invest the time to learn shortcuts. They rapidly lose patience with long menu sequences and multiple dialog boxes when they know exactly what they want to do. Therefore, shortcut keys reduce the number of interactions

- ✓ For a given task. Also, designers should provide macro facilities for users to create their own shortcuts.

3) Offer Informative Feedback

- ✓ Every action a user takes should result in some type of feedback from the computer so the user knows that the action was recognized.

- ✓ Even keyboard clicks help the user, so an electronic "click" is included deliberately by the operating system. If the user clicks a button, the button should visually change and perhaps make a sound.

4) Design Dialogs to Yield Closure

- ✓ Each dialog with the system should be organized with a clear sequence—a beginning, middle, and end.

- ✓ Any well-defined task has a beginning, middle, and end, so users' tasks on the computer should also feel this way. If the user is thinking, "I want to check my messages," as in the earlier manager and assistant dialog example, the dialog begins with a request, exchanges information, and then ends.

- ✓ The user can get lost if it is not clear when a task starts and ends. In addition, the user often focuses intently on a task, so when it is confirmed that the task is complete, the user can clear his or her mind and get ready to focus on the next task.

5) Offer Simple Error Handling

- ✓ The systems designer must prevent the user from making errors whenever possible. A chief way to do this is to limit available options and allow the user to choose from valid options at any point in the dialog.

- ✓ Adequate feedback, as discussed previously, also helps reduce errors.
- ✓ For example, if the user typed in an invalid customer ID, the system should tell the user that and then place the insertion point in the customer ID text box with the previously typed number displayed and ready to edit.

- ✓ This way, the user can see the mistake and edit it rather than having to retype the entire ID.

6) Permit Easy Reversal of Actions

- ✓ Users need to feel that they can explore options and take actions that can be cancelled or reversed without difficulty. This is one way that users learn about the system—by experimenting.

- ✓ It is also a way to prevent errors; as users recognize they have made a mistake, they cancel the action.

7) Support Internal Locus of Control

- ✓ Experienced users want to feel that they are in charge of the system and that the system responds to their commands.
- ✓ They should not be forced to do anything or made to feel as if the system is controlling them. Systems should make users feel that they are deciding what to do. Designers can provide much of this comfort and control through the wording of prompts and messages. Writing out a dialog like the manager and assistant message dialog given previously will lead to a design that conveys the feeling of control.

8) Reduce Short-Term Memory Load

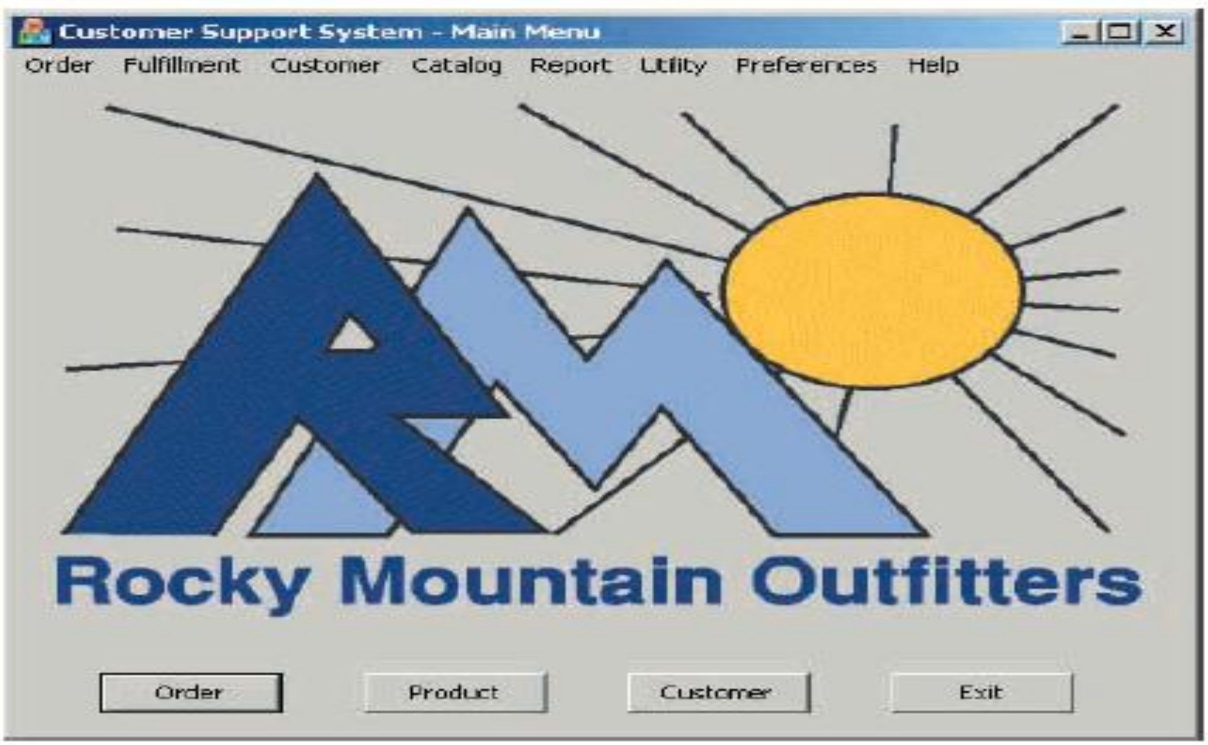
- ✓ People have many limitations, and short-term memory is one of the biggest. People can remember only about seven chunks of information at a time.

- ✓ The interface designer cannot assume that the user will remember anything from form to form, or dialog box to dialog box, during an interaction with the system. If the user has to stop and ask, "Now what was the filename? The customer ID? The product description?", then the design places too much of a burden on the user's memory.

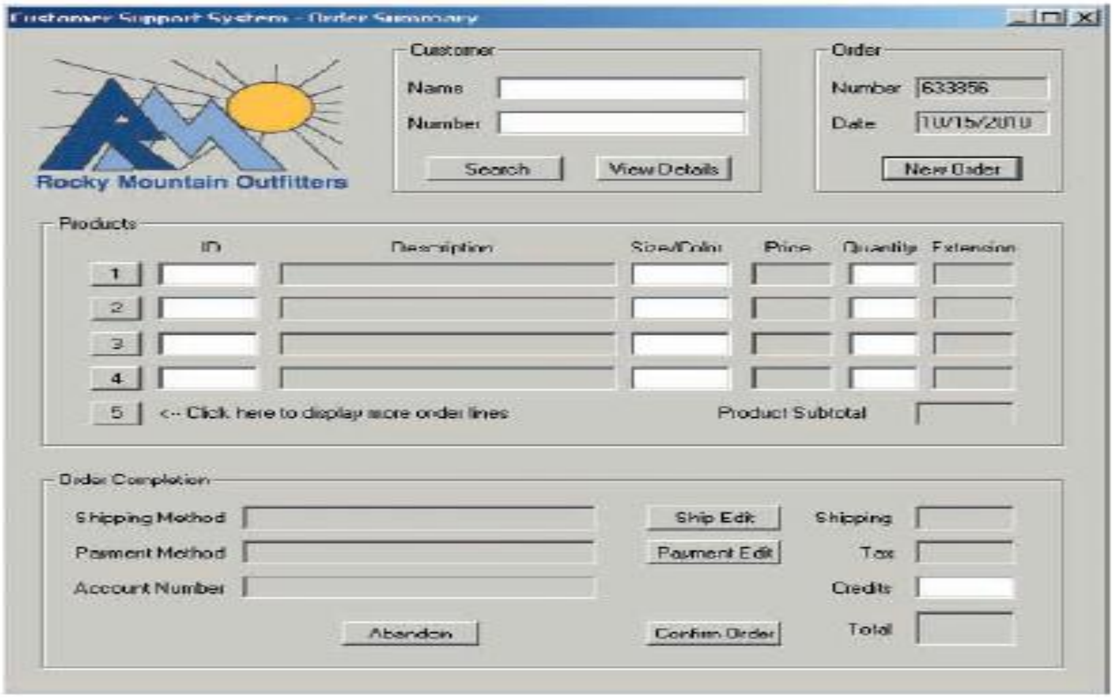
- **Examples of UI**

Home Page

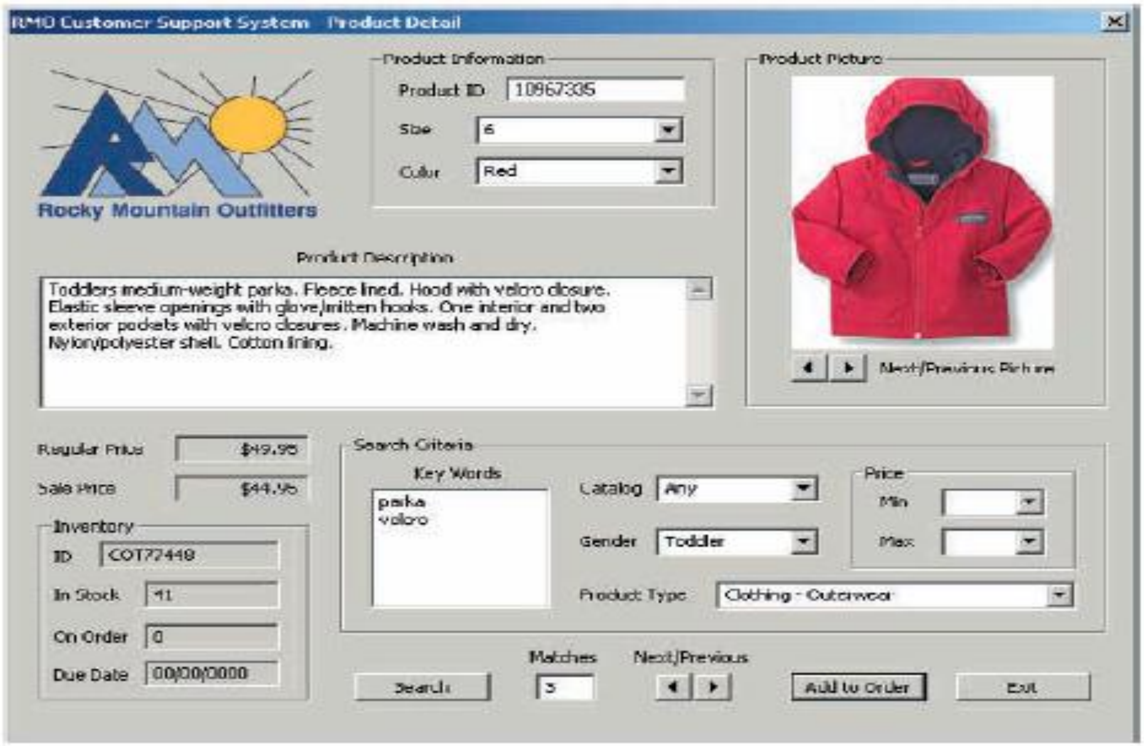
Main Menu Form



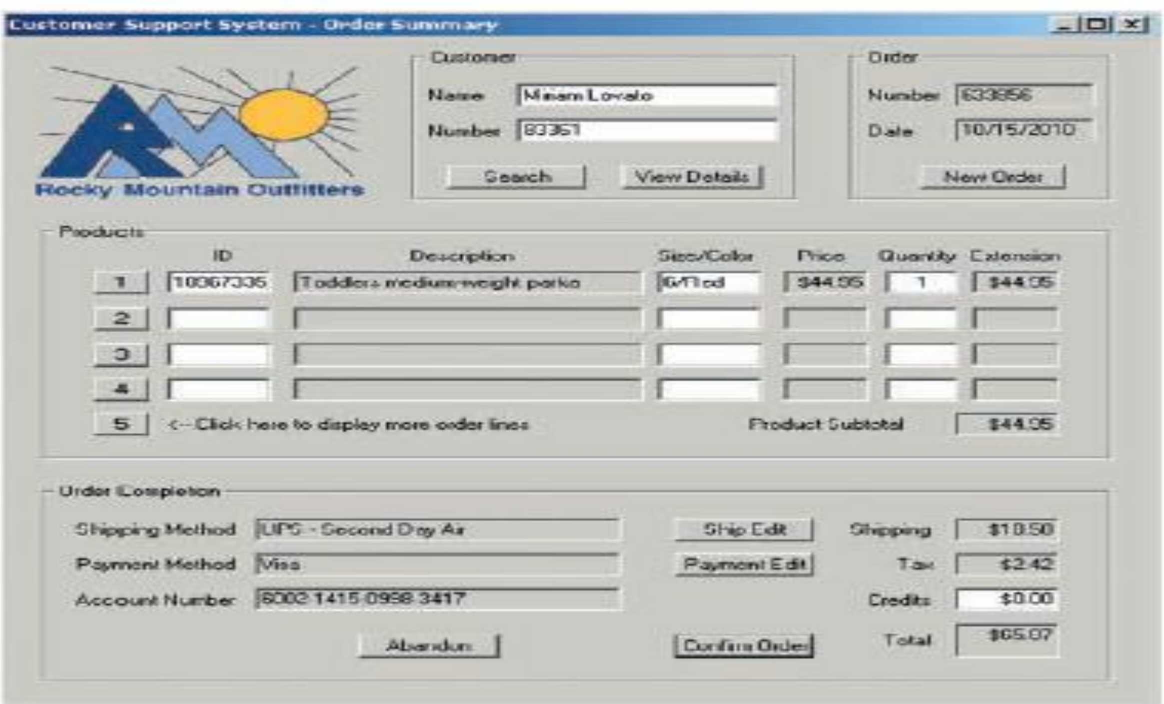
Order Summary Form beginning with new order



Product Detail form after User has search for product



Order Summary form after user adds the product



Shipping and Payment option form for the completed order