Chapter 4

Software Design

Software Design

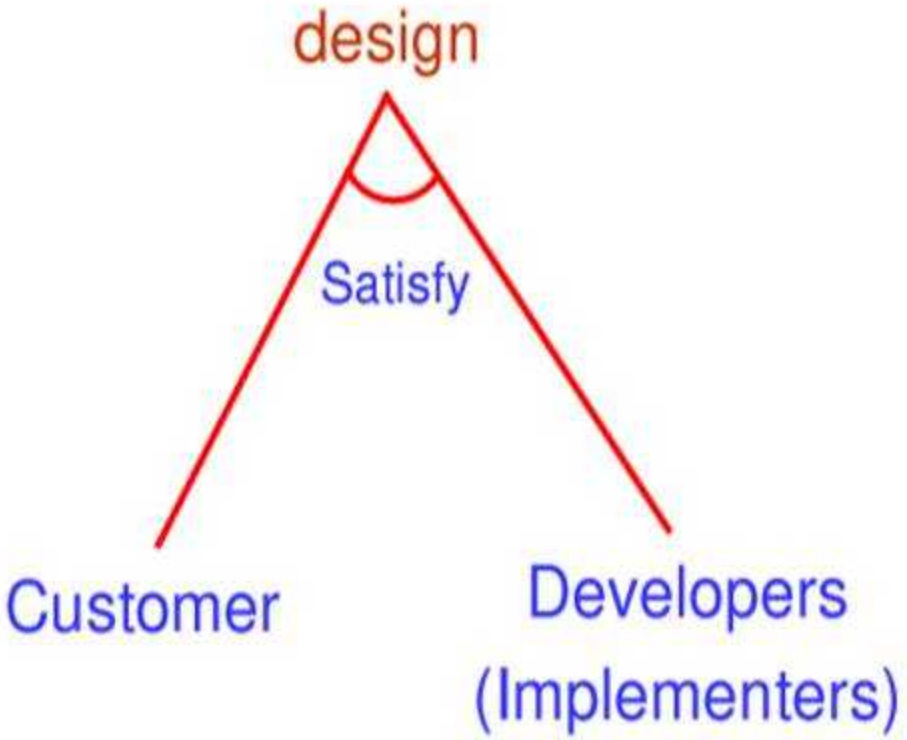
- More creative than analysis
- Problem solving activity

WHAT IS DESIGN

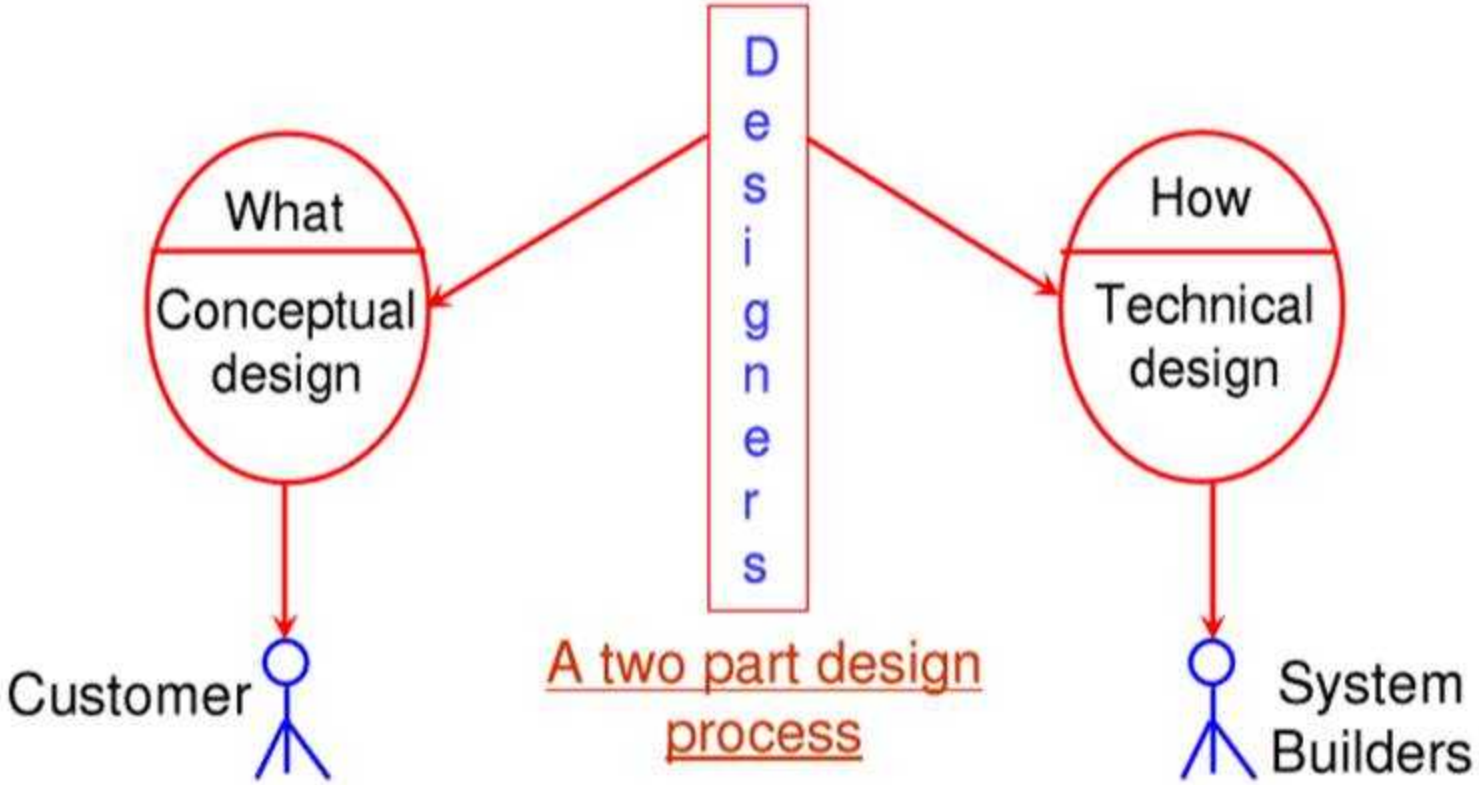
'HOW'

Software design document (SDD)

Software Design



Conceptual Design and Technical Design



Software Design

Conceptual design answers :

- ✓ Where will the data come from ?
- ✓ What will happen to data in the system?
- ✓ How will the system look to users?
- ✓ What choices will be offered to users?
- ✓ What is the timings of events?
- ✓ How will the reports & screens look like?

Software Design

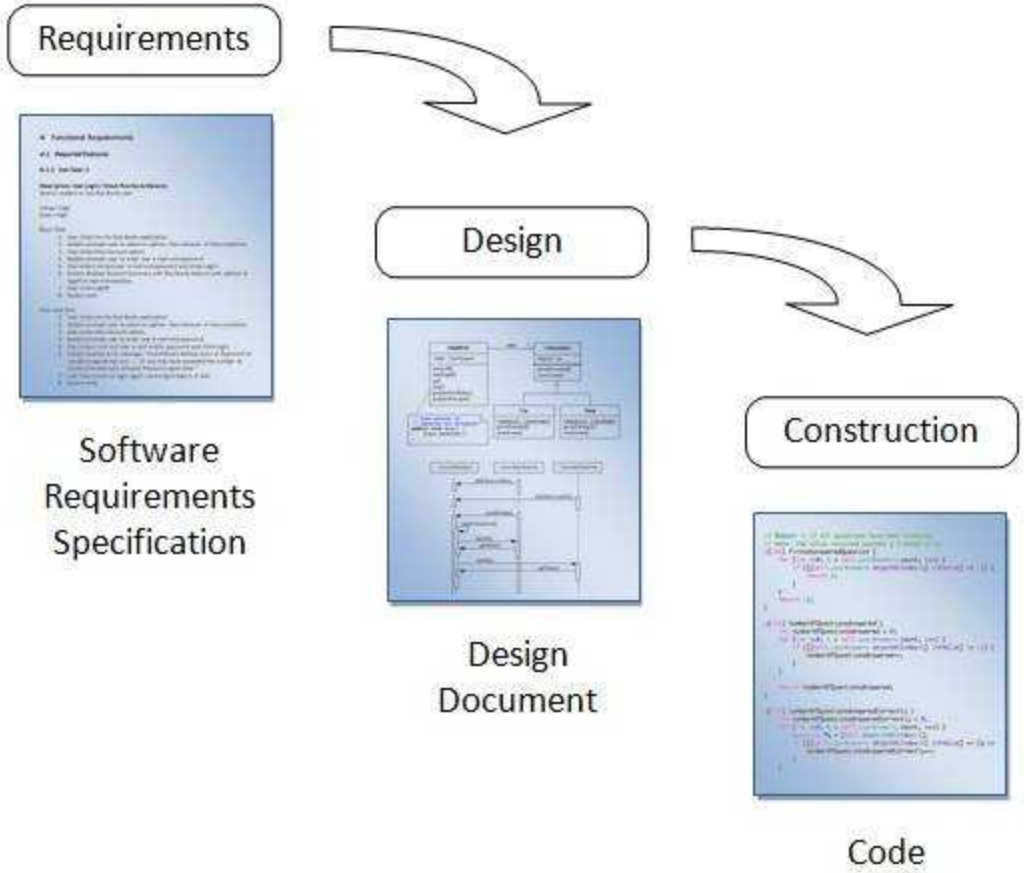
Technical design describes :

- Hardware configuration
- Software needs
- Communication interfaces
- I/O of the system
- Software architecture
- Network architecture
- Any other thing that translates the requirements in to a solution to the customer's problem.

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

What is Software Design?

- Design bridges that gap between knowing what is needed (software requirements specification) to entering the code that makes it work (the construction phase).
- Design is both a verb and a noun.

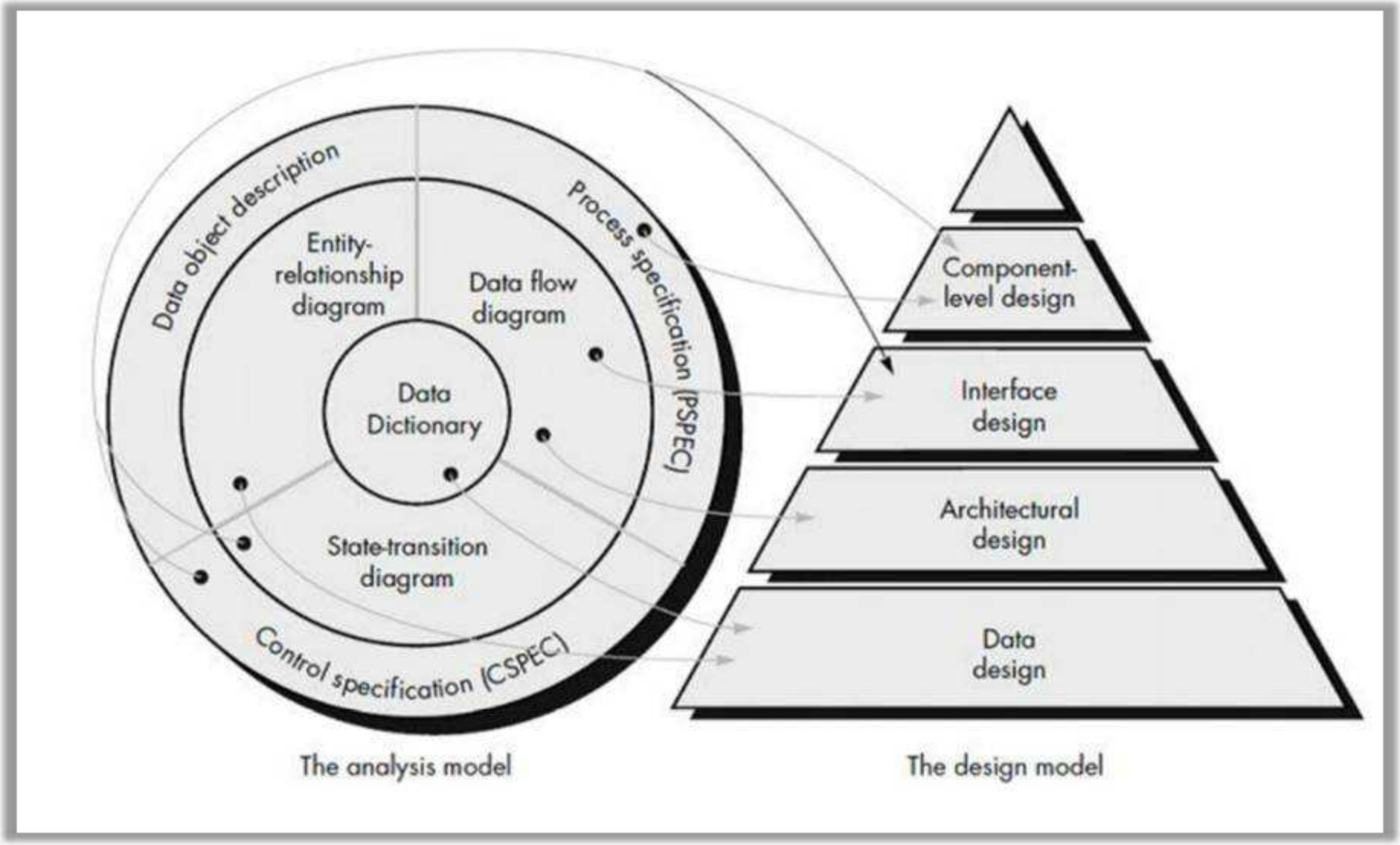


Design Within The Context of Software Engineering

- Software design sits at the technical kernel of software engineering and is applied regardless of the software process model being used. **Beginning once software requirements have been analyzed and modeled, software design is the first of three technical activities-designs, code generation, and test that are required to build and verify the software.**
- Each activity transforms information in a manner that ultimately results in validated computer software.

During the design process *the software requirements model is transformed into design models that describe the details of the data structures, system architecture, interface, and components. Each design product is reviewed for quality before moving to the next phase of software development.*

Translating the Analysis model into the Design model



Design Specification Models

- **Data design** - created by transforming the analysis information model (data dictionary and ERD) into data structures required to implement the software
- **Architectural design** - defines the relationships among the major structural elements of the software, it is derived from the system specification, the analysis model, and the subsystem interactions defined in the analysis model (DFD)
- **Interface design** - describes how the software elements communicate with each other, with other systems, and with human users; the data flow and control flow diagrams provide much the necessary information
- **Component-level design** - created by transforming the structural elements defined by the software architecture into procedural descriptions of software components using information obtained from the PSPEC, CSPEC, and STD

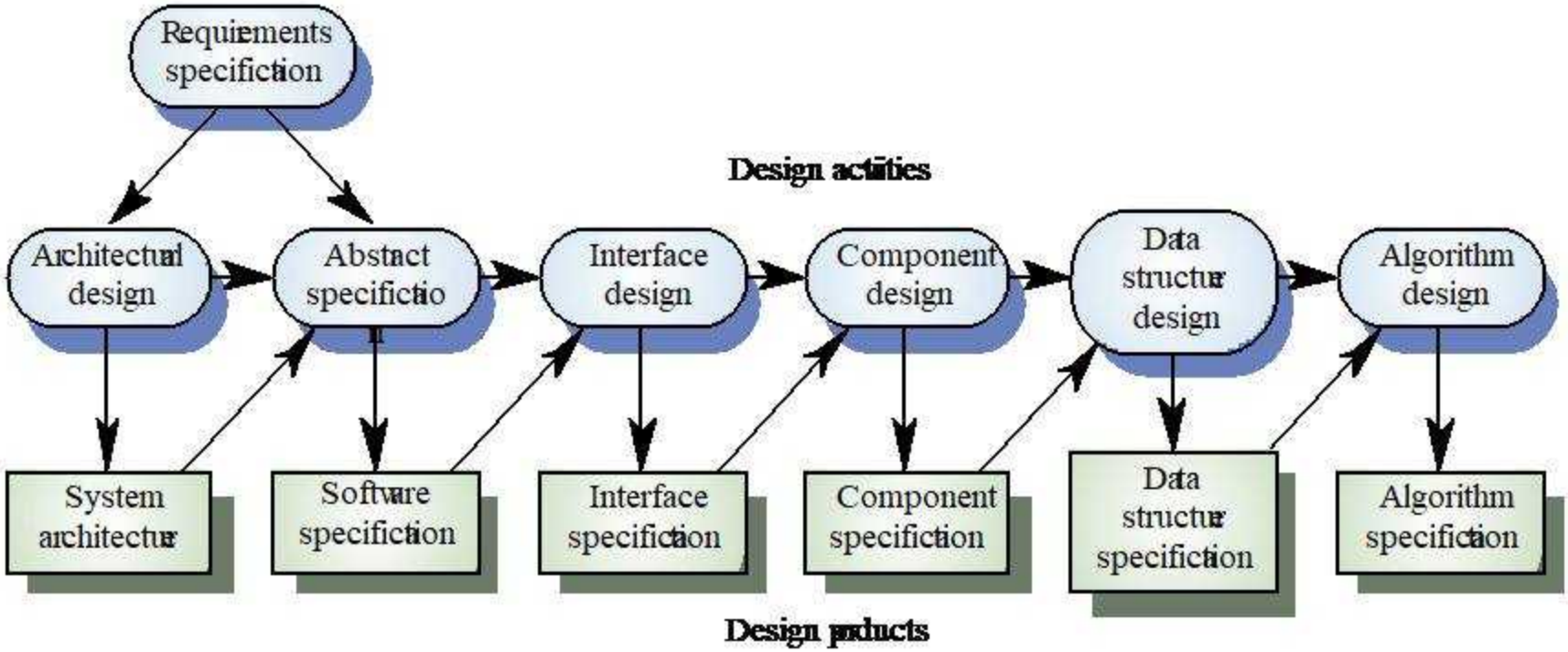
Three Characteristics that serve as a guide for the evaluation of a good design

- ***The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.***
- ***The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.***
- ***The design should provide a complete picture of the software, addressing the data, functional and behavioral domains from an implementation perspectives.***

Design and Quality Goals

- The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
- The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
- The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

Phases in the Design Process



Design Phases

Architectural design: Identify sub-systems.

Abstract specification: Specify sub-systems.

Interface design: Describe sub-system interfaces.

Component design: Decompose sub-systems into components.

Data structure design: Design data structures to hold problem data.

Algorithm design: Design algorithms for problem functions.

Design Guidelines

A design should

- exhibit good architectural structure
- be modular
- contain distinct representations of data, architecture, interfaces, and components (modules)
- lead to data structures that are appropriate for the objects to be implemented and be drawn from recognizable design patterns
- lead to components that exhibit independent functional characteristics
- lead to interfaces that reduce the complexity of connections between modules and with the external environment
- Design should be represented using a notation that effectively communicates its meaning.

How to achieve the Quality

- A design should exhibit an architecture that (1) has been created using recognizable architectural styles or patterns, (2) is composed of components that exhibit good design characteristics and (3) can be implemented in an evolutionary fashion
  - For smaller systems, design can sometimes be developed linearly.
- A design should be modular; that is, the software should be logically partitioned into elements or subsystems
- A design should contain distinct representations of data, architecture, interfaces, and components.
- A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
- A design should lead to components that exhibit independent functional characteristics.
- A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
- A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
- A design should be represented using a notation that effectively communicates its meaning.

Design Principles

The design

- should not suffer from "tunnel vision“
- should be traceable to the analysis model
- should not reinvent the wheel
- should "minimize intellectual distance" between the software and the problem as it exists in the real world should exhibit uniformity and integration
- should be structured to accommodate change
- should be structured to degrade gently, even with bad data, events, or operating conditions are encountered
- should be assessed for quality as it is being created
- should be reviewed to minimize conceptual (semantic) errors

Quality Attributes: The **FURPS** quality attributes represent a target for all software design

- Functionality: is assessed by evaluating the feature set and capabilities of the program, the generality of the functions that are delivered.
- Usability: is assessed by considering human factors. Overall aesthetics, consistency and documentation.
- Reliability: is evaluated by measuring the frequency and severity of failure, the accuracy of output results, the ability to recover from failure.
- Performance: is measured by processing speed, response time, resource consumption, throughput and efficiency.
- Supportability: combines the ability to extend the program (extensibility), adaptability, serviceability – these 3 attributes represent a more common term Maintainability.

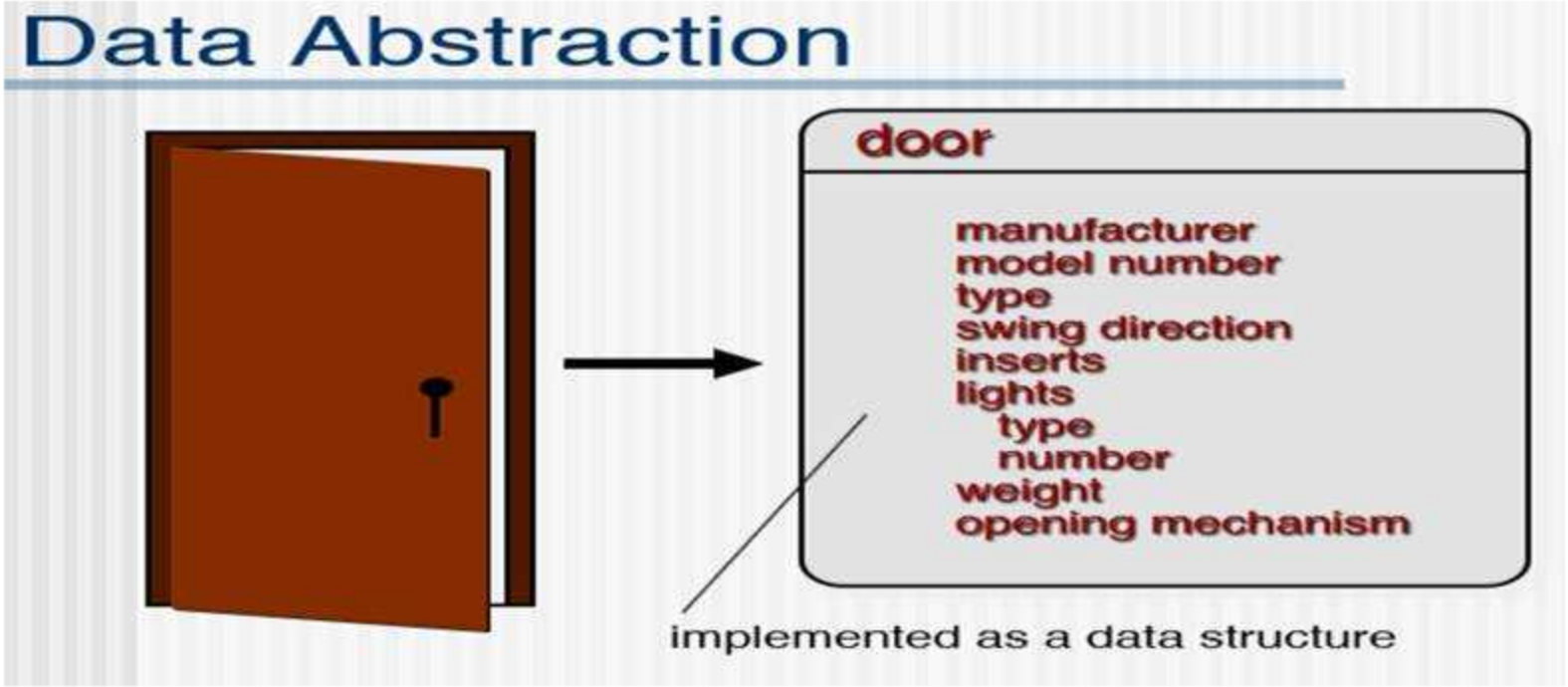
Fundamental Concepts in Design

- Abstraction—data, procedure, control
- Architecture—the overall structure of the software
- Patterns—"conveys the essence" of a proven design solution
- Separation of concerns—any complex problem can be more easily handled if it is subdivided into pieces
- Modularity—manifestation of separation of concerns
- Information Hiding—controlled interfaces, no details of algorithms/data
- Functional independence—single-minded function and low coupling
- Refinement—elaboration of detail for all abstractions
- Aspects—a mechanism for understanding how global requirements affect design
- Refactoring—a reorganization technique that simplifies the design
- OO design concepts—Appendix II
- Design Classes—provide design detail that will enable analysis classes to be implemented

Design Concepts

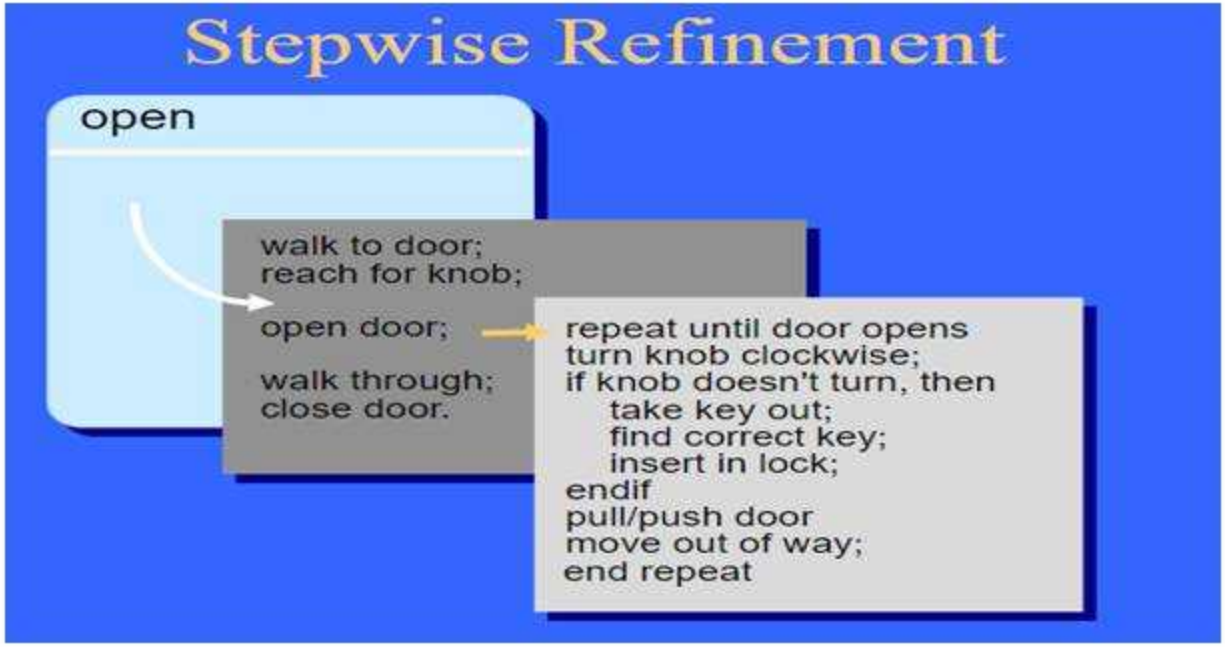
Abstraction

- Abstraction is one of the fundamental ways that we as humans cope with complexity.
- Allows designers to focus on solving a problem without being concerned about irreverent lower levels.



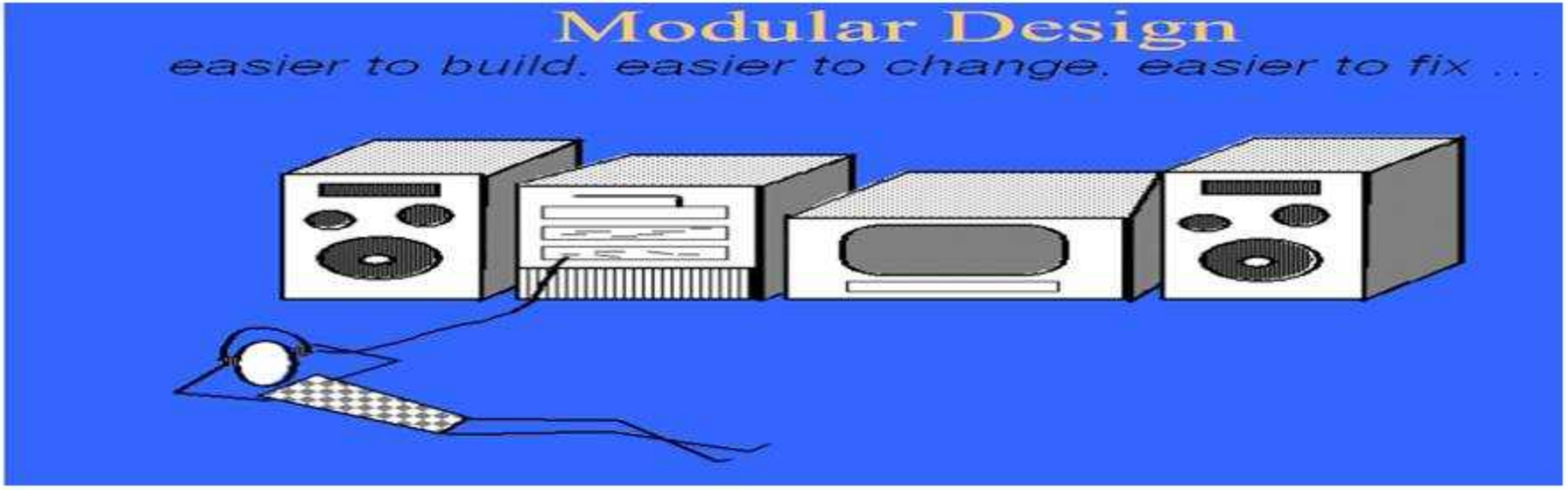
Refinement

- Process of elaboration where the designer provides successively more details for each design component.
- Refinement is thus a process where one or several instructions of the program are decomposed into more detailed instructions. Stepwise refinement is a top down strategy –Basic architecture is developed iteratively –Step wise hierarchy is developed Forces a designer to develop low level details as the design progresses –Design decisions at each stage .



Modularity

- The degree to which software can be understood by examining its components independently of one another.
- ***Modularity is the single attribute of software that allows a program to be intellectually manageable.***
- ***It enhances design clarity, which in turn eases implementation, debugging, testing, documenting, and maintenance of software product.***
- In almost all instances, you should break the design into many modules, hoping to make understanding easier and as a consequence, reduce the cost required to build the software.



Functional Independence

- Functional independence is achieved by developing modules with "single-minded" function and an "aversion" to excessive interaction with other modules.
- Software with effective modularity, that is, independent modules, is easier to develop because function may be compartmentalized and interfaces are simplified.
- Individual modules are easier to maintain (and test).
- There are many advantages of Modularization in software engineering. Some of these are given below:
  - ✓ Easy to understand the system.
  - ✓ System maintenance is easy.
  - ✓ A module can be used many times as their requirements. No need to write it again and again
- Conclusively, functional independence is a key to design, and design is the key to software quality.
- Independence is assessed using two criteria: Cohesion and Coupling.
- Cohesion is an indication of the relative function of strength of a module.
- Coupling is an indication of the relative interdependence among modules.

Cohesion

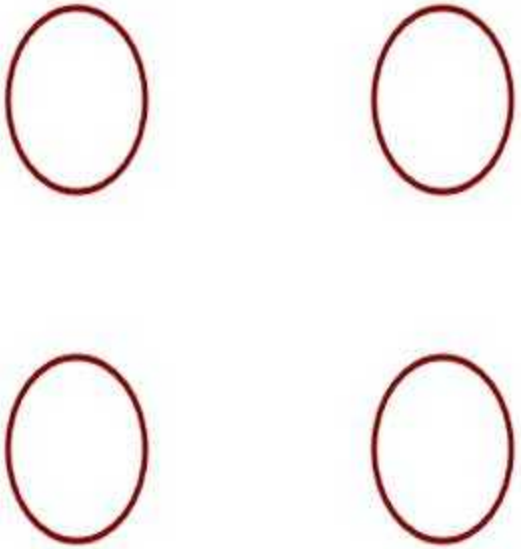
- Cohesion is the indication of the relationship within module.
- Cohesion shows the module's relative functional strength.
- Cohesion is a degree (quality) to which a component / module focuses on the single thing.
- While designing you should strive for high cohesion i.e. a cohesive component/ module focus on a single task (i.e., single-mindedness) with little interaction with other modules of the system.
- Cohesion is the kind of natural extension of data hiding for example, class having all members visible with a package having default visibility. Cohesion is Intra – Module Concept.

Coupling

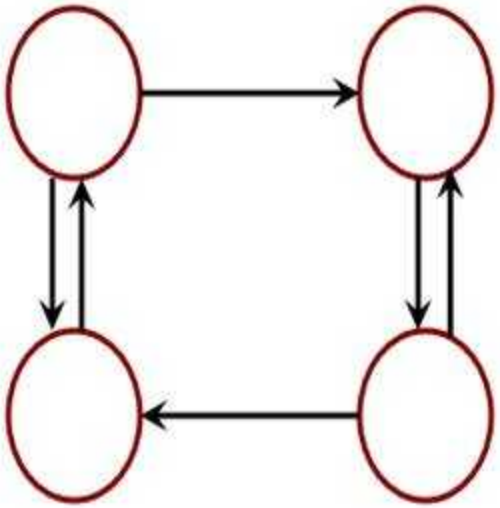
- Coupling is the indication of the relationships between modules.
- Coupling shows the relative dependence/interdependence among the modules.
- Coupling is a degree to which a component / module is connected to the other modules.
- While designing you should strive for low coupling i.e. dependency between modules should be less
- Making private fields, private methods and non public classes provides loose coupling.
- Coupling is Inter -Module Concept

Module Coupling

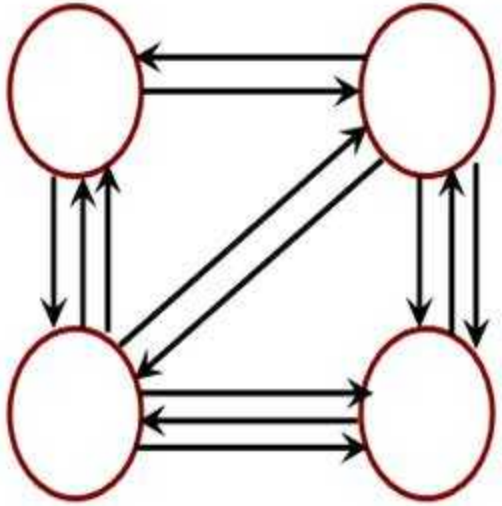
Coupling is the measure of the degree of interdependence between modules.



(Uncoupled : no dependencies)



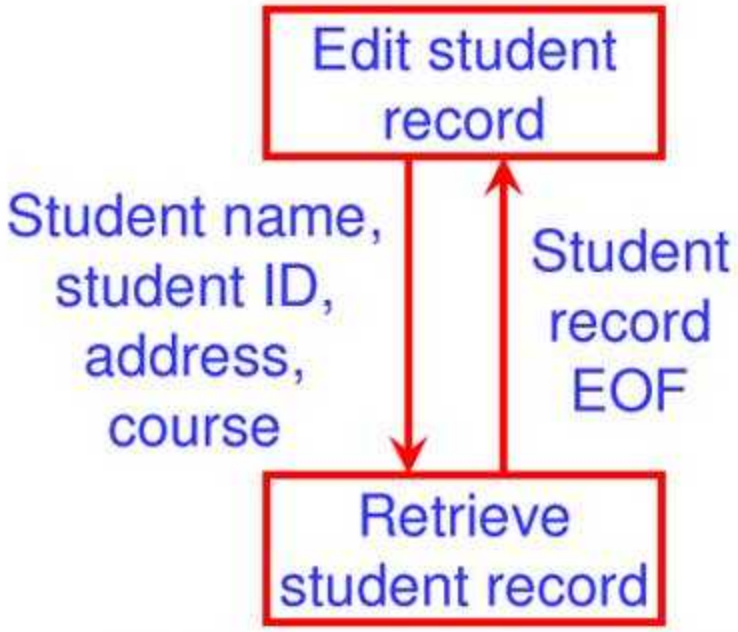
Loosely coupled:
some dependencies



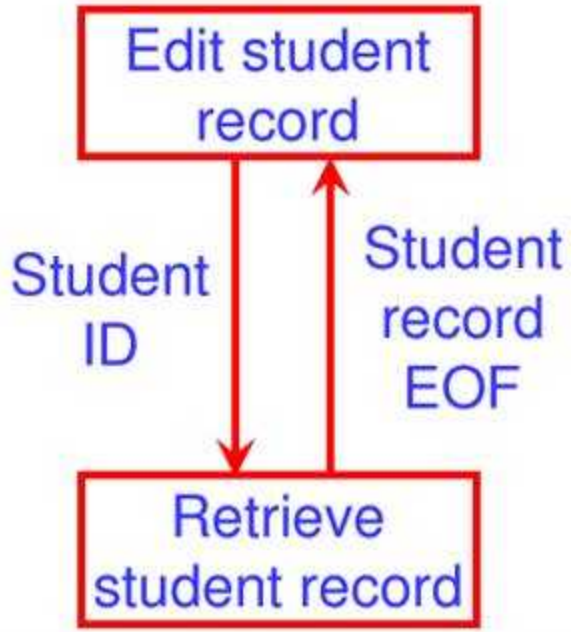
Highly coupled:
many dependencies

Fig: Module Coupling

Consider the example of editing a student record in a 'student information system'.

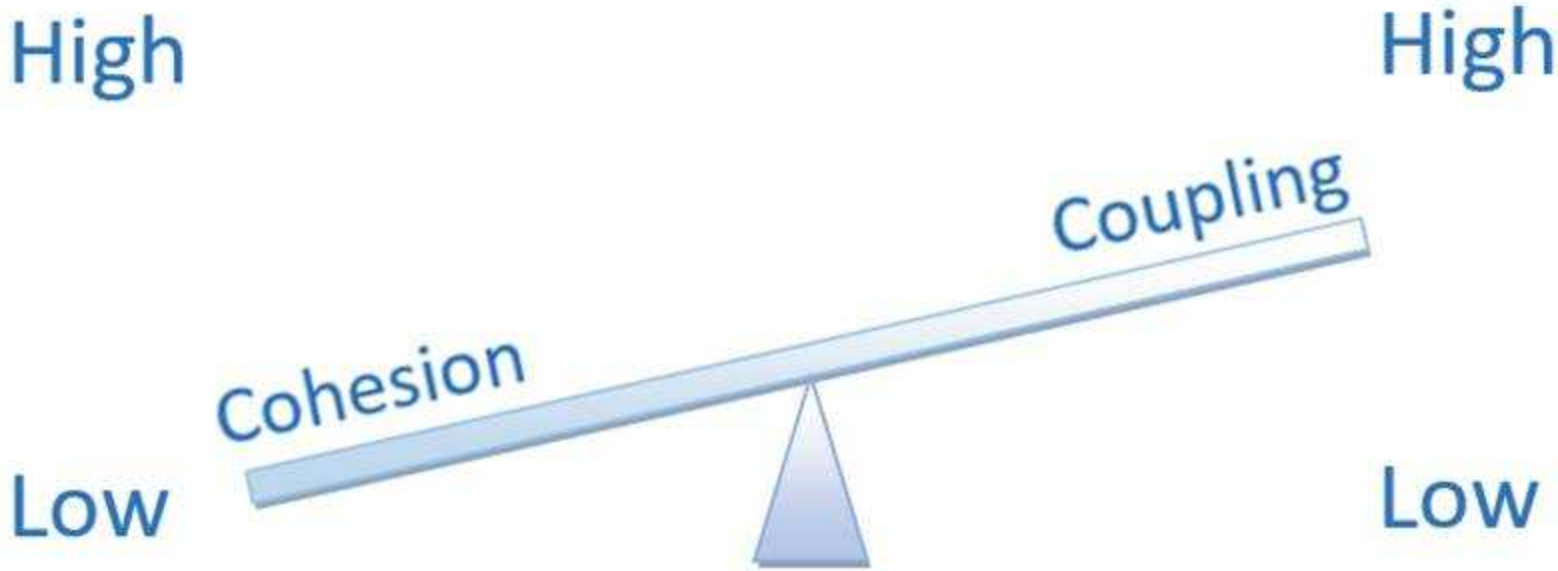


Poor design: Tight Coupling



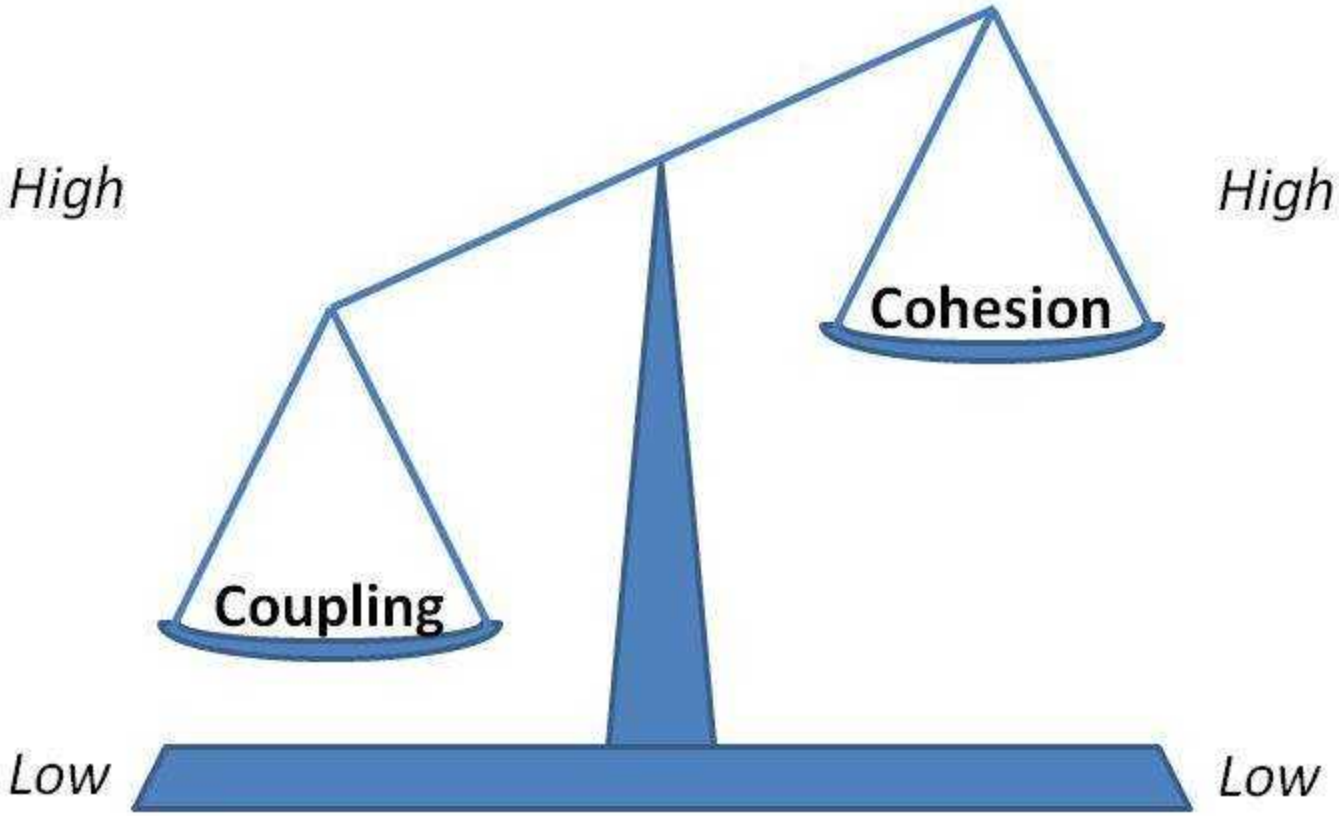
Good design: Loose Coupling

Coupling and Cohesion Tend to be Inversely Correlated



When Coupling is high, cohesion tends to be low and vise versa.

Relationship between Coupling and Cohesion



Software Design

Types of Coupling

Functional Cohesion	Best (high)
Sequential Cohesion	
Communicational Cohesion	
Procedural Cohesion	
Temporal Cohesion	
Logical Cohesion	
Coincidental Cohesion	Worst (low)

Types of Module Cohesion/Strength

Types of Coupling

Assignment

Types of Cohesion

Assignment

Refactoring

- Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure.
- Refactoring consists of improving the internal structure of an existing program's source code, while preserving its external behavior.
- It is a disciplined way to clean up code that minimizes the chances of introducing bugs. In essence when you refactor you are improving the design of the code after it has been written.
- It makes a software system easier to understand and cheaper to modify without changing its observable behavior by changing its internal structure.
- **The purposes of refactoring according to M. Fowler are stated in the following:**
  - ✓Refactoring Improves the Design of Software
  - ✓Refactoring Makes Software Easier to Understand
  - ✓Refactoring Helps Finding Bugs
  - ✓Refactoring Helps Programming Faster

Patterns

- Design pattern can be used throughout software design. One analysis model has been developed, the designer can examine a detailed representation of the problem to be solved. A design pattern is a reusable solution to a commonly occurring design problem.
- Design patterns are adapted for the unique characteristics of the particular problem
- Just as there are levels of design, there are levels of design patterns:

  - **Architecture Styles/Patterns:** These patterns define the overall structure of the software, indicate the relationships among subsystems and software components.

  - **Design Patterns:** these patterns address a specific element of the design such as an aggregation of components to solve some design problem, relationships among components or the mechanisms for effecting component-to- component communication.

  - **Programming Idioms:** These language-specific patterns generally implement an algorithmic element of a component , a specific interface protocol, or a mechanism for communication among components.

Information Hiding

- The intent of information hiding is to hide the details of data structures and procedural processing behind a module interface. Knowledge of details need not be known by users of the module.
- he practice of hiding the details of a module with the goal of controlling access to the details from the rest of the system.
- Information hiding or data hiding in programming is about protecting data or information from any inadvertent change throughout the program. Information hiding is a powerful OOP feature. Information hiding is closely associated with encapsulation.
- Information or data hiding is a concept which protects the data from direct modification by other parts of the program.
- The feature of information hiding is applied using Class in most of the programming languages.

Architecture

- The software architecture of a program or computing system is the structure or structures of the system which comprise
  - The software components
  - The externally visible properties of those components
  - The relationships among the components
- Software architectural design represents the structure of the data and program components that are required to build a computer-based system.

Why Architecture?

Software Architecture is a representation that enables a software engineer to:

(1) **analyze the effectiveness of the design** in meeting its stated requirements,

(2) **consider architectural alternatives** at a stage when making design changes is still relatively easy, and

(3) **reduce the risks** associated with the construction of the software.

Why is Architecture Important?

- **Representations of software architecture are an enabler** for communication between all parties (stakeholders) interested in the development of a computer-based system.
- **The architecture highlights early design decisions** that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
- **Architecture "constitutes a relatively small, intellectually graspable model** of how the system is structured and how its components work together" .
- **Large Scale Reuse** The architecture may be reusable across a range of systems.

Architectural Design

- Architectural Design represents the structure of data and program components that are required to build a computer based systems. ***It considers the architectural style that the system will take, the structure and properties of the components that constitute the system, and the interrelationships that occur among all architectural components of a system***.
- ***The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is architectural design***
- ***The output of this design process is a description of the software architecture.***
- ***It involves identifying major system components and their communications***

Architectural Design Process

- Basic Steps
  - Creation of the data design
  - Derivation of one or more representations of the architectural structure of the system
  - Analysis of alternative architectural styles to choose the one best suited to customer requirements and quality attributes
  - Elaboration of the architecture based on the selected architectural style
- A database designer creates the data architecture for a system to represent the data components
- A system architect selects an appropriate architectural style derived during system engineering and software requirements analysis.

Architectural design process

- **System structuring**
  - The system is decomposed into several principal sub-systems and communications between these sub-systems are identified
- **Control modelling**
  - A model of the control relationships between the different parts of the system is established
- **Modular decomposition**
  - The identified sub-systems are decomposed into modules

Architecture attributes

- **Performance**
  - Localise operations to minimise sub-system communication
- **Security**
  - Use a layered architecture with critical assets in inner layers
- **Safety**
  - Isolate safety-critical components
- **Availability**
  - Include redundant components in the architecture
- **Maintainability**
  - Use fine-grain, self-contained components

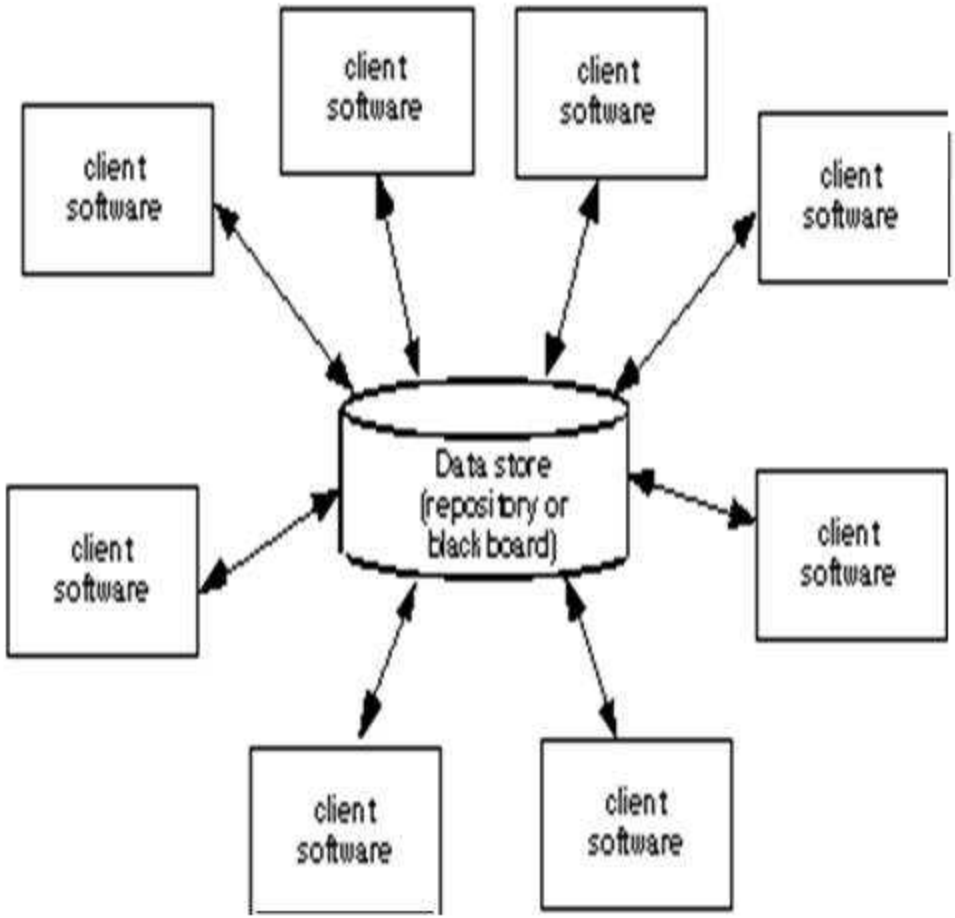
Architectural Styles

Each style describes a system category that encompasses: (1) a **set of components** (e.g., a database, computational modules) that perform a function required by a system, (2) a **set of connectors** that enable "communication, coordination and cooperation" among components, (3) **constraints** that define how components can be integrated to form the system, and (4) **semantic models** that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

- **Data-centered architectures**
- **Data flow architectures**
- **Call and return architectures**
- **Object-oriented architectures**
- **Layered architectures**

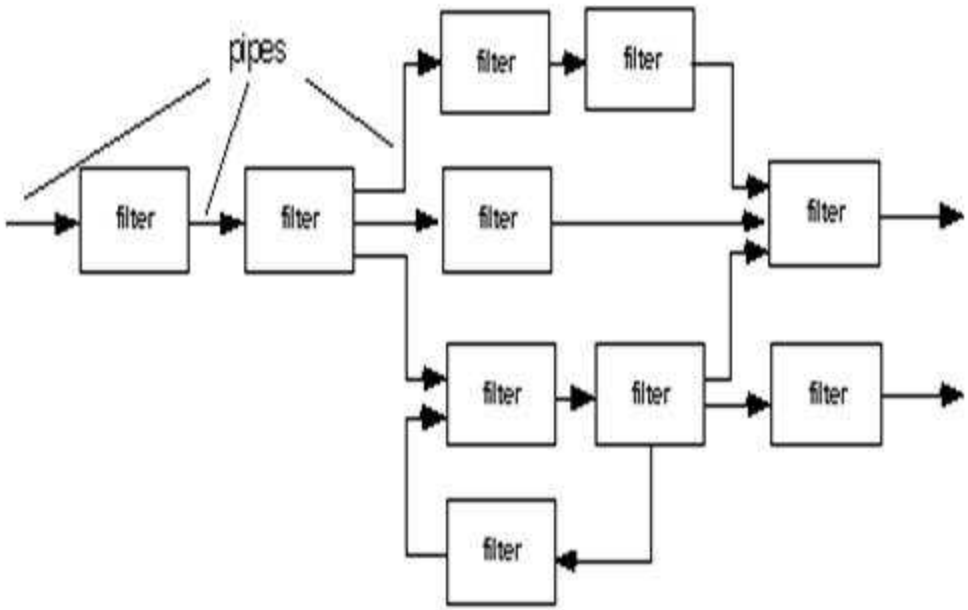
Data Centered Architecture

A data store (e.g. a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, retrieve or delete data within the store. i.e. Client software accesses a central repository.



Data Flow Architecture

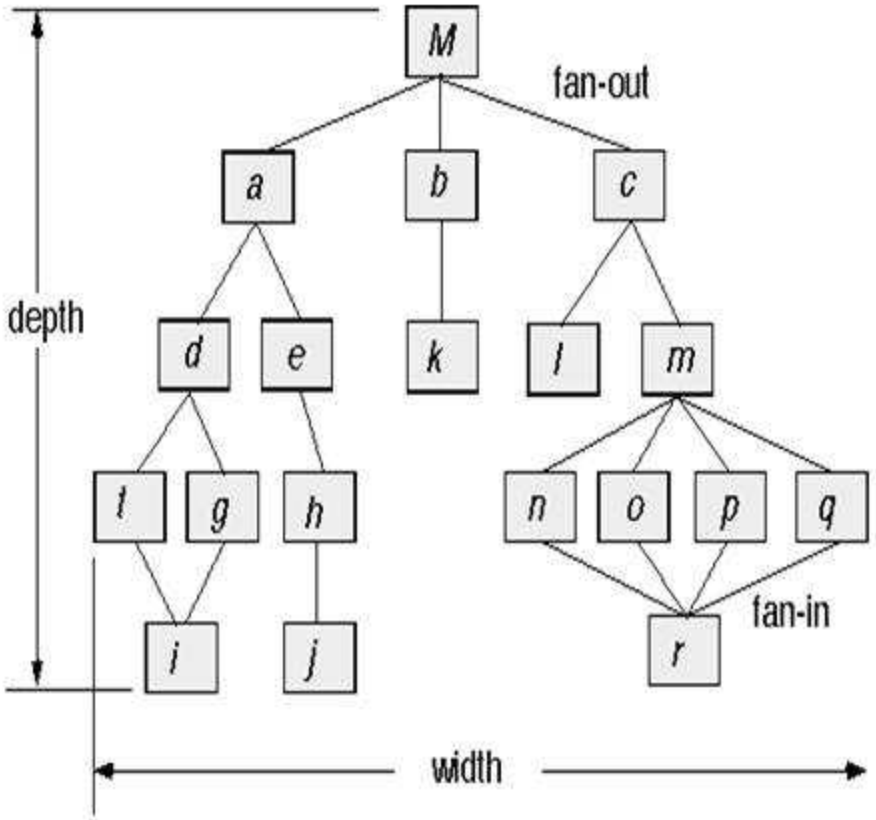
This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data. A pipe and filter structure has a set of components called filters connected by pipes that transmit data from one component to the next. Each processing step is enclosed within a **filter** component. Data to be processed is passed through **pipes** Each filter works independently of those components. The filter doesn't require knowledge of the workings of its neighboring filters.



) pipes and filters

Call and Return Architecture

This architectural style enables a software designer to achieve a program structure that is relatively easy to modify and scale. The program structure decomposes function into a control hierarchy where a "main" program invokes a number of program components, which in turn may invoke still other components.

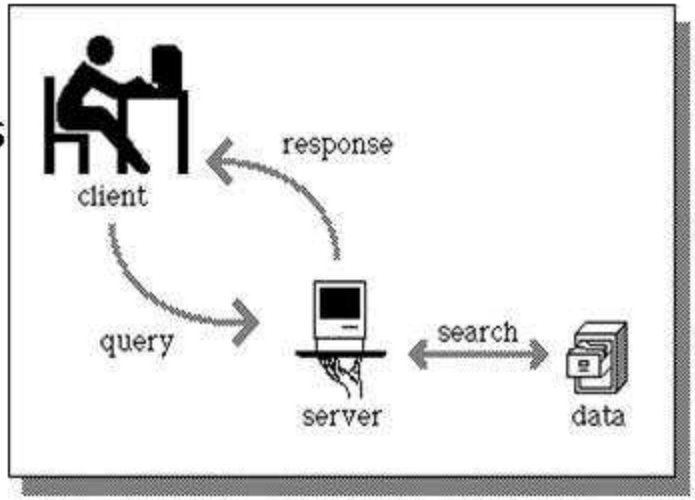


Object Oriented Architecture

- The properties of inheritance, polymorphism, encapsulation, and composition being provided by OOA come in handy in producing highly modular (highly cohesive and loosely coupled), usable and reusable software applications.
- The object-oriented style is suitable if we want to encapsulate logic and data together in reusable components. Also, the complex business logic that requires abstraction and dynamic behavior can effectively use this OOA. The components of a system encapsulate data and the operations that must be applied to manipulate the data. Communication and coordination between components is accomplished via message passing.

Client/server architecture

- This pattern segregates the system into two main applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures. This pattern helps to design ***distributed systems*** that involve a client system and a server system and a connecting network.
- Distributed system model which shows how data and processing is distributed across a range of components.
  - **Set of stand-alone servers** which provide specific services such as printing, data management, etc.
  - **Set of clients** which call on these services
  - **Network** which allows clients to access servers

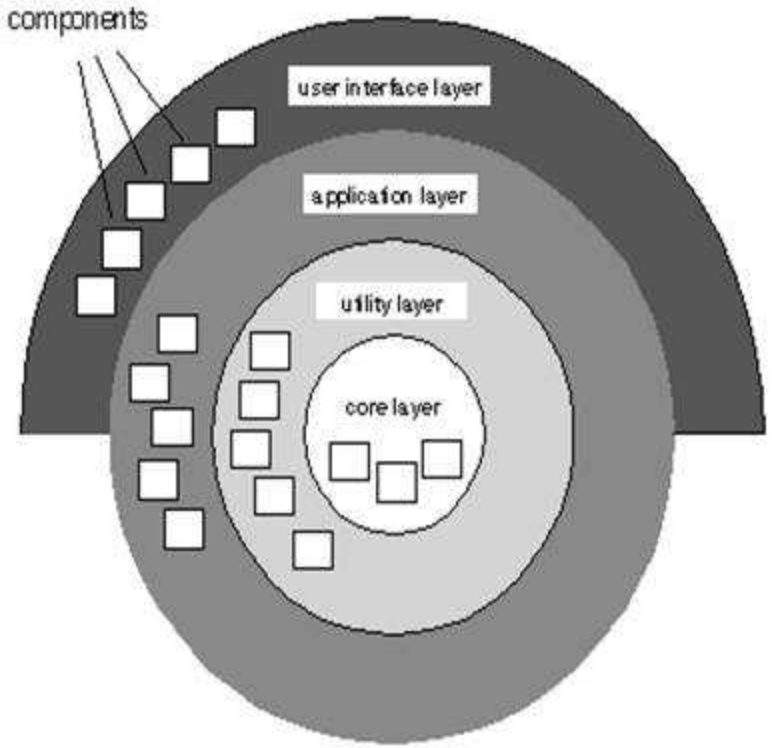


Layered Architecture

Organises the system into a set of layers (or abstract machines) each of which provide a set of services

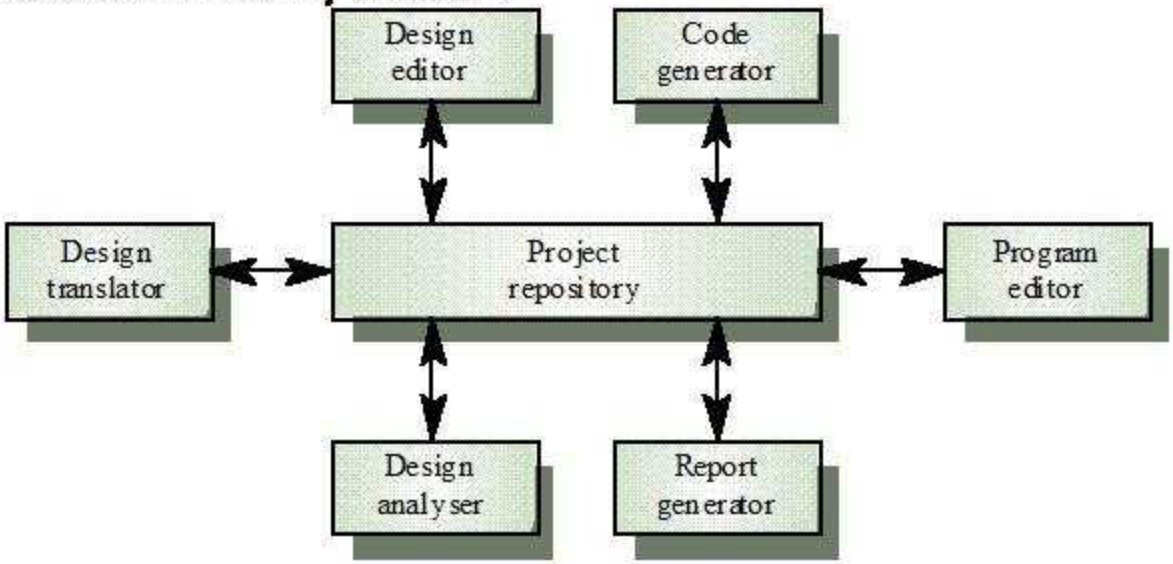
Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.

A number of different layers are defined, each accomplishing a particular operation. At the outer layer, components service user interface operations. At the inner layer, components perform operating system interfacing. Intermediate layers provide utility and application software solutions.



The repository model

- **Sub-systems must exchange data**. This may be done in two ways:
  - **Shared data is held in a central database or repository and may be accessed by all sub-systems**
  - **Each sub-system maintains its own database and passes data explicitly to other sub-systems**
- When large amounts of data are to be shared, the repository model of sharing is most commonly used .



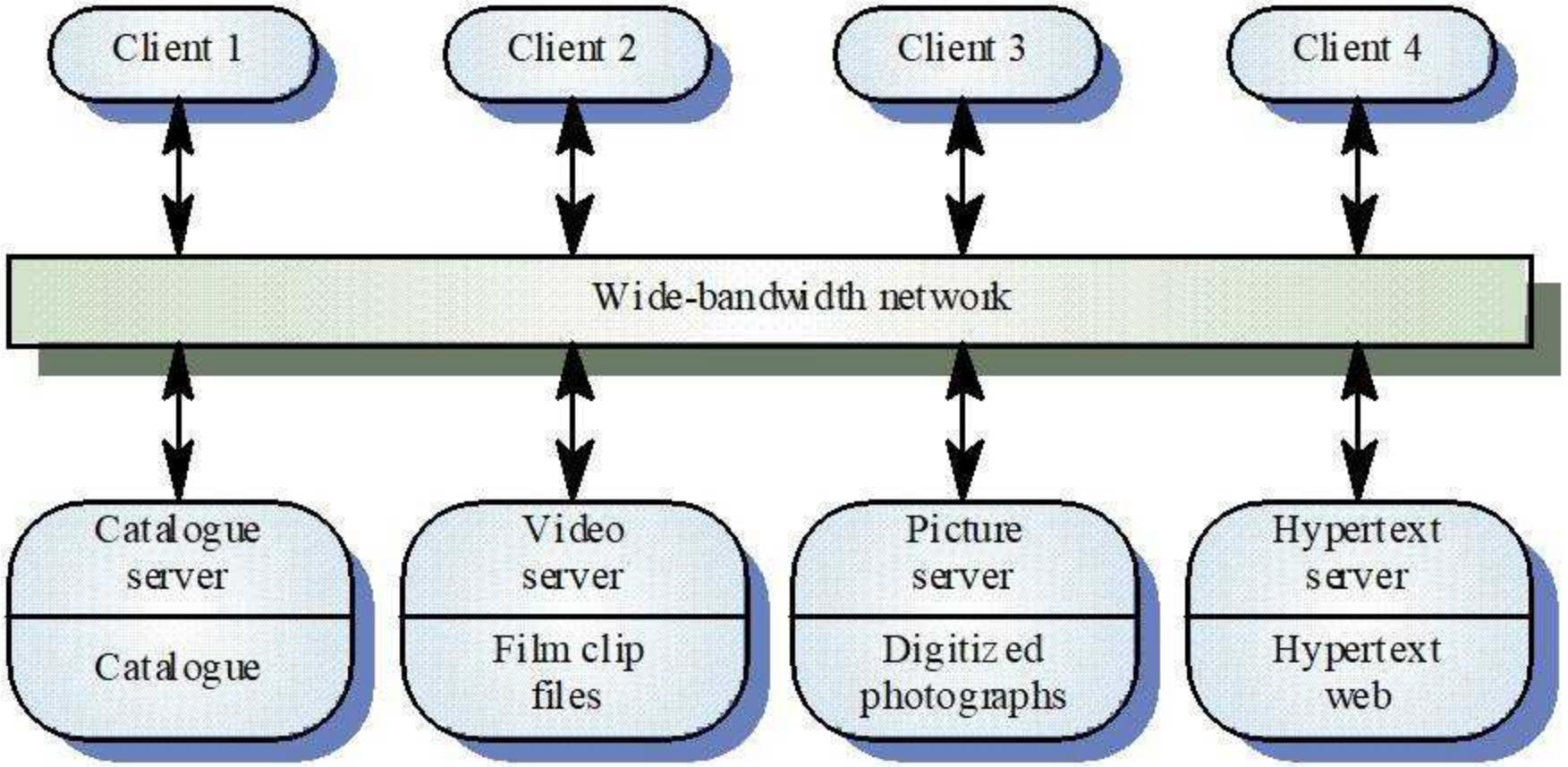
Repository model

- **Advantages**
  - Efficient way to share large amounts of data
  - Sub-systems need not be concerned with how data is produced
  - Centralised management e.g. backup, security, etc.
  - Sharing model is published as the repository schema
- **Disadvantages**
  - Sub-systems must agree on a repository data model. Inevitably a compromise
  - Data evolution is difficult and expensive
  - No scope for specific management policies
  - Difficult to distribute efficiently

The client-server model

- Distributed system model which shows how data and processing are distributed across a range of processors. (machines)
- Major components:
  - A set of **stand-alone servers** which provide specific services such as printing, file management, etc.
  - A set of **clients** which call on these services
  - A **network** which allows clients to access these services

Example of a simple client-server based system:
Film and picture library



Client-server model *advantages*

- Supports distributed computing
- Underlying network makes distribution of data straightforward.
- No shared data model so servers may organize data to optimize their performance.
- Distinction between servers and clients may allow use of cheaper hardware.
- Relatively easy to expand or upgrade system.

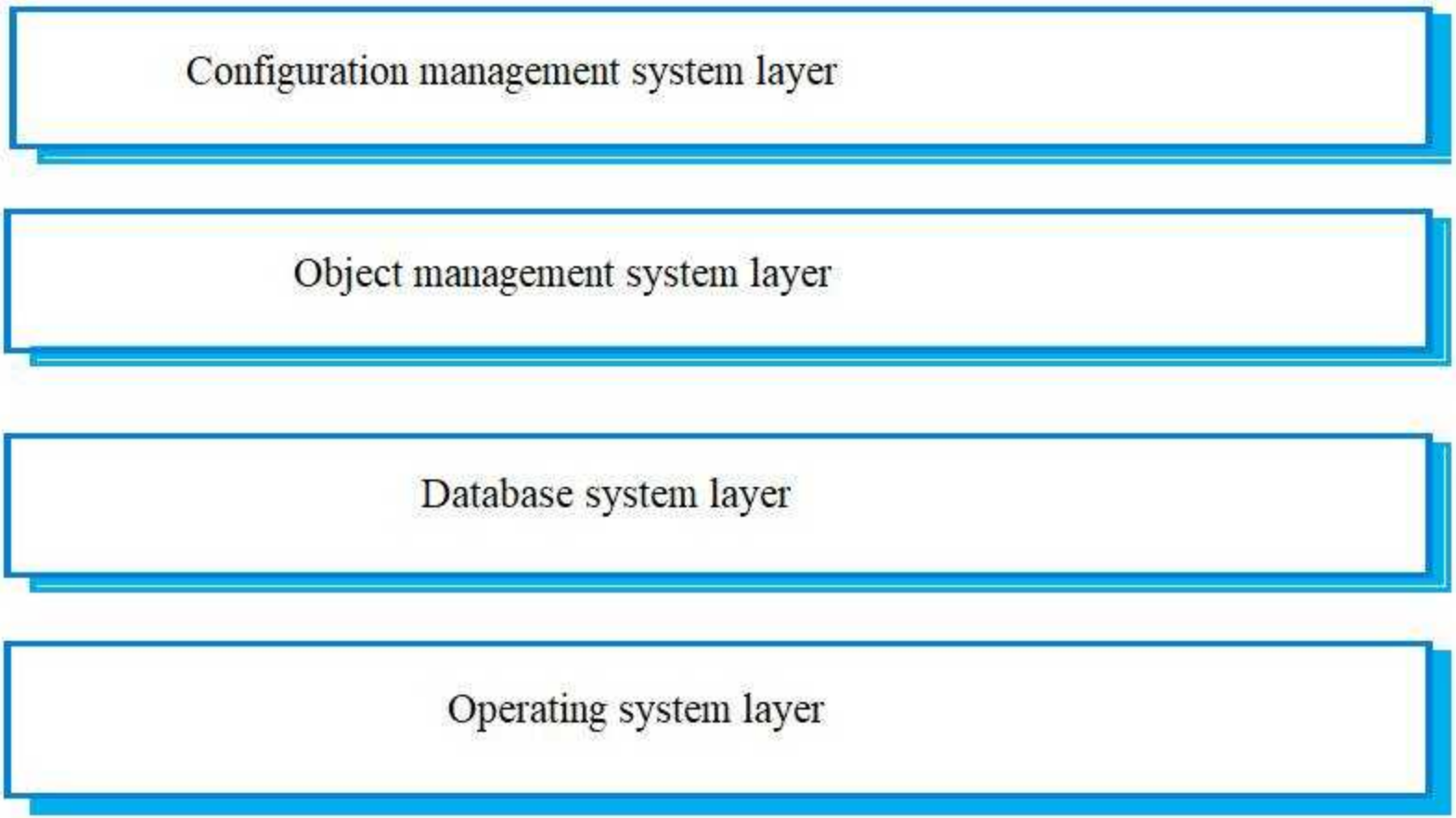
Client-server model *disadvantages*

- Relatively complex architecture – problem determination can be difficult.
- No shared data model so data integration may be problematic. (must be ad hoc)
- Redundant data management activities in each server, possibly. (Consider film and picture library.)
- No central register of names and services, so it may be hard to find out what servers and services are available.

The abstract machine (Layered) model

- Organizes a system into a series of **layers.**
- Each layer defines an abstract machine and provides a set of services used to implement the next level of abstract machine.
- When a layer interface changes, only the adjacent layer is affected.

Example of a simple abstract machine (Layered) Model: version management system



Modular decomposition styles

- Styles of decomposing sub-systems into modules.
- A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems.
- A module is a system component that provides services to other components but would not normally be considered as a separate system.

Modular decomposition

- Two modular decomposition models covered

1) **Object Oriented Decomposition**

   Where you decompose a system into a set of communication objects.

2) **Function Oriented Decomposition**

   Where you decompose a system into functional module that accept input data and transform it into output data.

Object models

- Object-oriented decomposition is concerned with identifying object classes, their attributes and operations.
- When implemented, objects are created from these classes and some control model used to coordinate object operations.

Note: Go through all the UML Diagrams done in my lecture class.

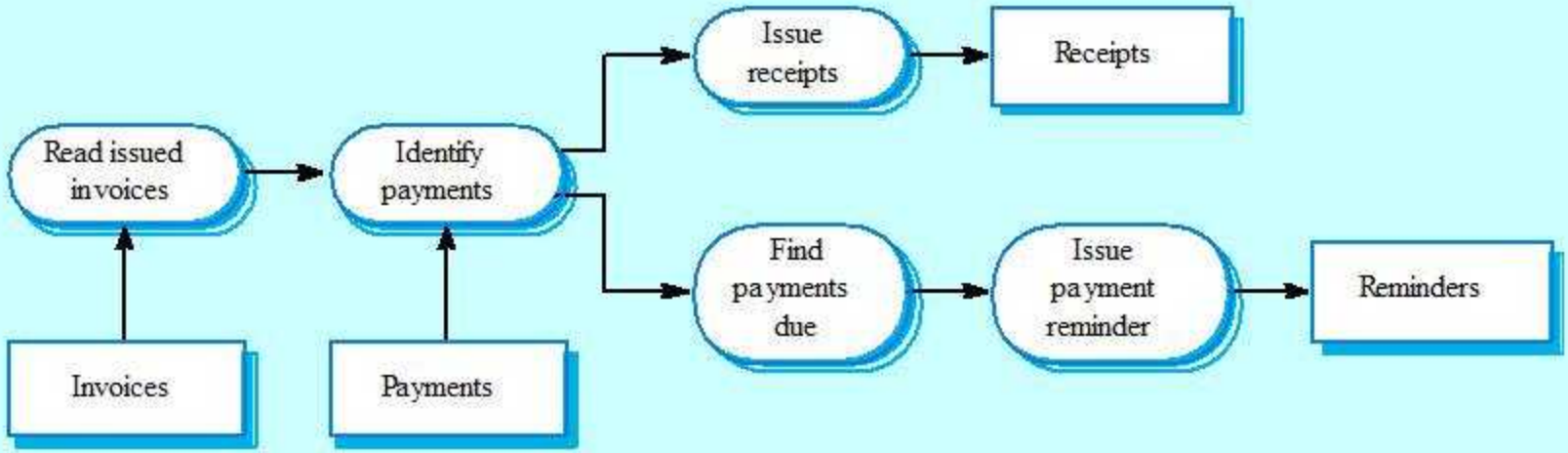
Object model advantages

- Objects are loosely coupled so their implementation can be modified without affecting other objects.
- The objects may reflect real-world entities.
- OO implementation languages are widely used.

Function-oriented pipelining

- Functional transformations process their inputs to produce outputs.
- May be referred to as a pipe and filter model

Invoice processing system



Pipeline model advantages

- Supports transformation reuse.
- Intuitive organisation for stakeholder communication.
- Easy to add new transformations.
- Relatively simple to implement as either a concurrent or sequential system.

User Interface Design

User Interface Design

- UI design is not just about the arrangement of media on a screen
- It's designing an entire experience for people, hence a "look and feel"
- Psychology: building a mental model
- Ergonomics: facilitating navigation

The user interface

- System users often judge a system by its interface rather than its functionality
- A poorly designed interface can cause a user to make catastrophic errors
- Poor user interface design is the reason why so many software systems are never used
- Most users of business systems interact with these systems through graphical user interfaces (GUIs)
- In some cases, legacy text-based interfaces are still used

GUI characteristics

Characteristic	Description
Windows	Multiple windows allow different information to be displayed simultaneously on the user's screen.
Icons	Icons different types of information. On some systems, icons represent files; on others, icons represent 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.

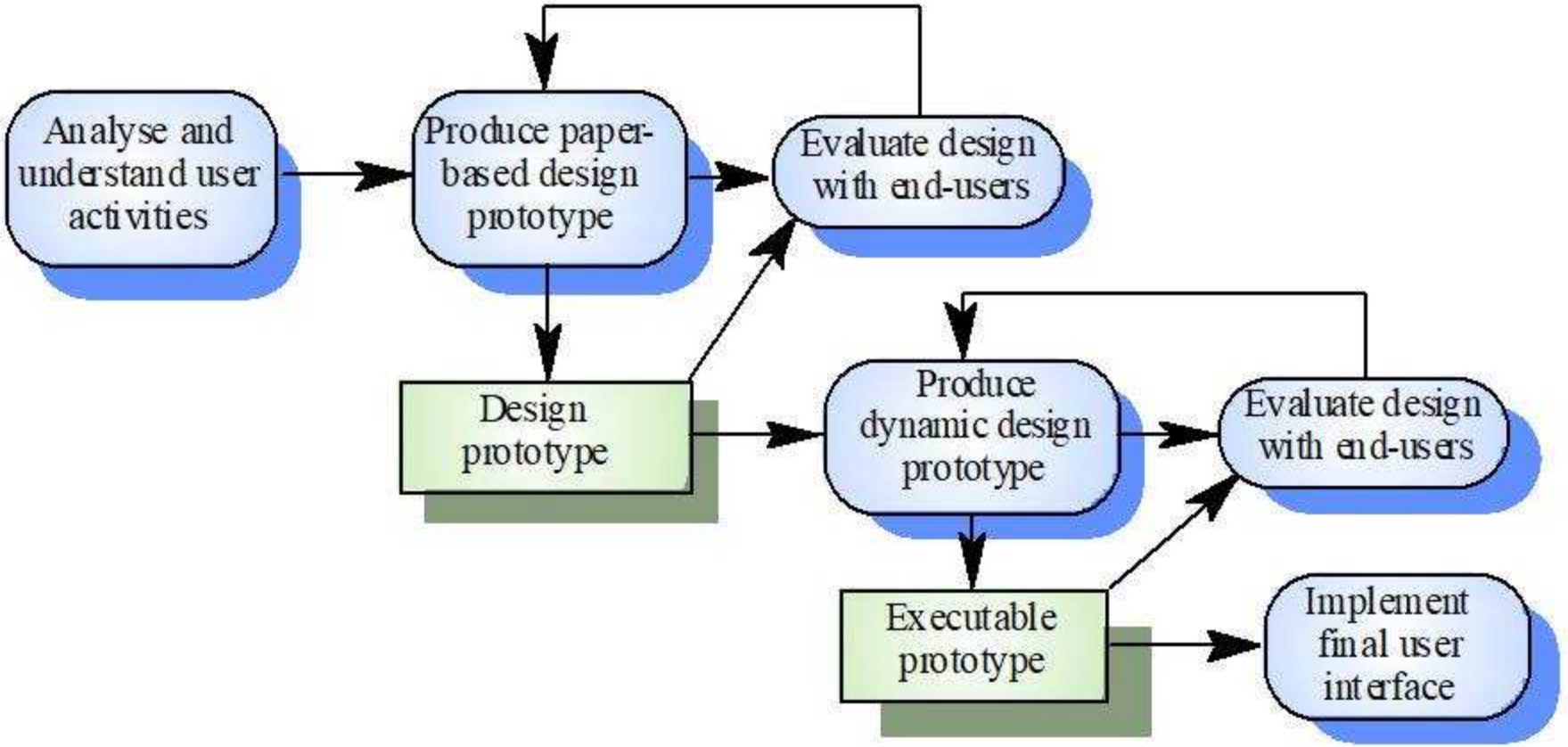
UI and UX

- The user interface is the graphical layout of an application. It consists of the buttons users click on, the text they read, the images, sliders, text entry fields, and all the rest of the items the user interacts with. This includes screen layout, transitions, interface animations and every single micro-interaction. Any sort of visual element, interaction, or animation must all be designed. UI designers are graphic designers. They're concerned with aesthetics.
- ***User experience (UX) is determined by how easy or difficult it is to interact with the user interface elements that the UI designers have created.***
- ***UI designers are tasked with deciding how the user interface will look, UX designers are in charge of determining how the user interface operates.***
- ***UX designer decides how the user interface works while the UI designer decides how the user interface looks.***

- UX design is the process used to determine what the experience will be like when a user interacts with your product.
- User experience design is an approach to design that takes the user into account
- UX is improving how useful, easy, pleasant, marketable, or addictive it is to use a product
- **UX design is a commitment to building products with the customer in mind**
- **UX design is so much more than just designing for a screen**

- *A successful user interface should be intuitive (not require training to operate), efficient (not create additional or unnecessary friction) and user-friendly (be enjoyable to use).*

User interface design process



User Interface

User interface is the front-end application view to which user interacts in order to use the software. The software becomes more popular if its user interface is:

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

UI design principles

- User familiarity
  - The interface should be based on user-oriented terms and concepts rather than computer concepts
  - E.g., an office system should use concepts such as letters, documents, folders etc. rather than directories, file identifiers, etc.
- Consistency
  - The system should display an appropriate level of consistency
  - Commands and menus should have the same format, command punctuation should be similar, etc.
- Minimal surprise
  - If a command operates in a known way, the user should be able to predict the operation of comparable commands.

UI design principles (cont.)

- Recoverability
  - The system should provide some resilience to user errors and allow the user to recover from errors
  - This might include an undo facility, confirmation of destructive actions, deletes, etc.
- User guidance
  - Some user guidance such as help systems, on-line manuals, etc. should be supplied
- User diversity
  - Interaction facilities for different types of user should be supported
  - E.g., some users have seeing difficulties and so larger text should be available
- Speak user's language
  - understandable instructions, feedback, error messages
- Anticipation
  - hide or grey out inactive features

User-system interaction

- Two problems must be addressed in interactive systems design
  - **How should information from the user be provided to the computer system?**
  - **How should information from the computer system be presented to the user?**

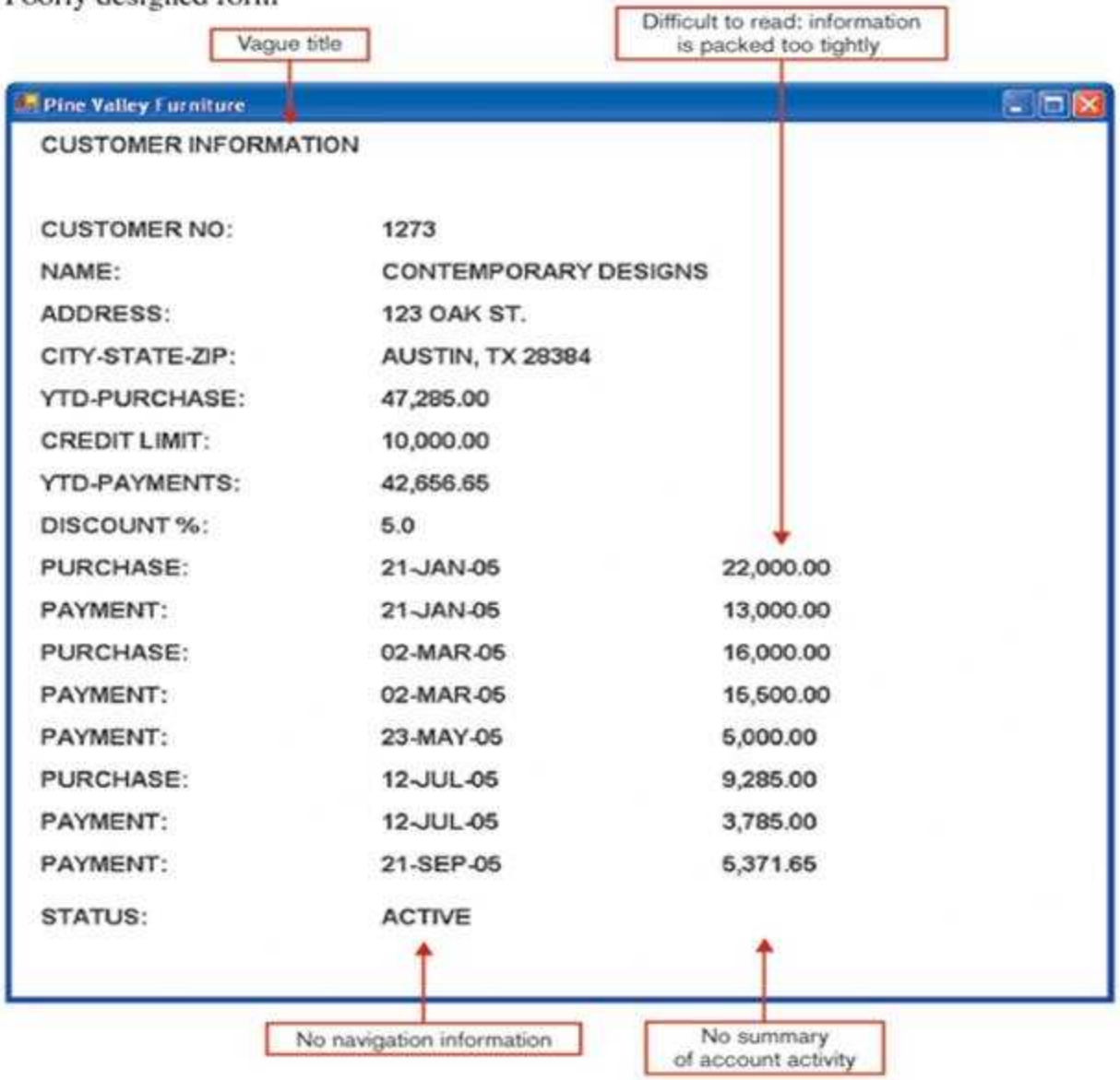
The various types of user interfaces include:

- graphical user interface (GUI)
- command line interface (CLI)
- menu-driven user interface
- touch user interface
- voice user interface (VUI)
- form-based user interface
- Natural language user interface

Guidelines for Form and Report Design

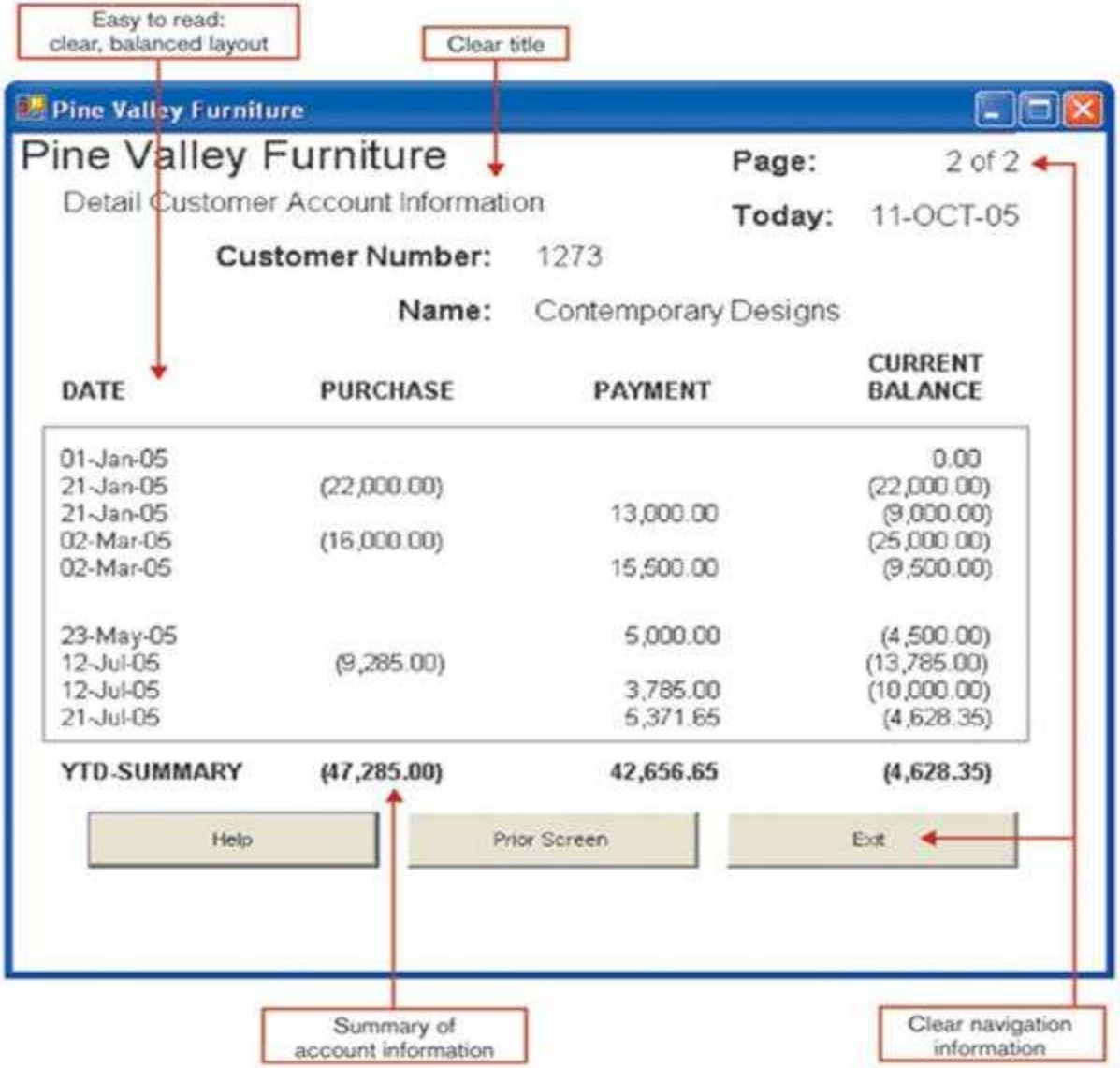
- **Meaningful titles**
  - clear, specific, version information, current date
- **Meaningful information**
  - include only necessary information
- **Balanced layout**
  - adequate spacing, margins, and clear labels
- **Easy navigation system**
  - show how to move forward and backward, and where you are currently

Figure 11-5a Contrasting customer information forms (Pine Valley Furniture) - Poorly designed form



A poor form design

Figure 11-5b Contrasting customer information forms (Pine Valley Furniture) - Improved design for form



A better form design

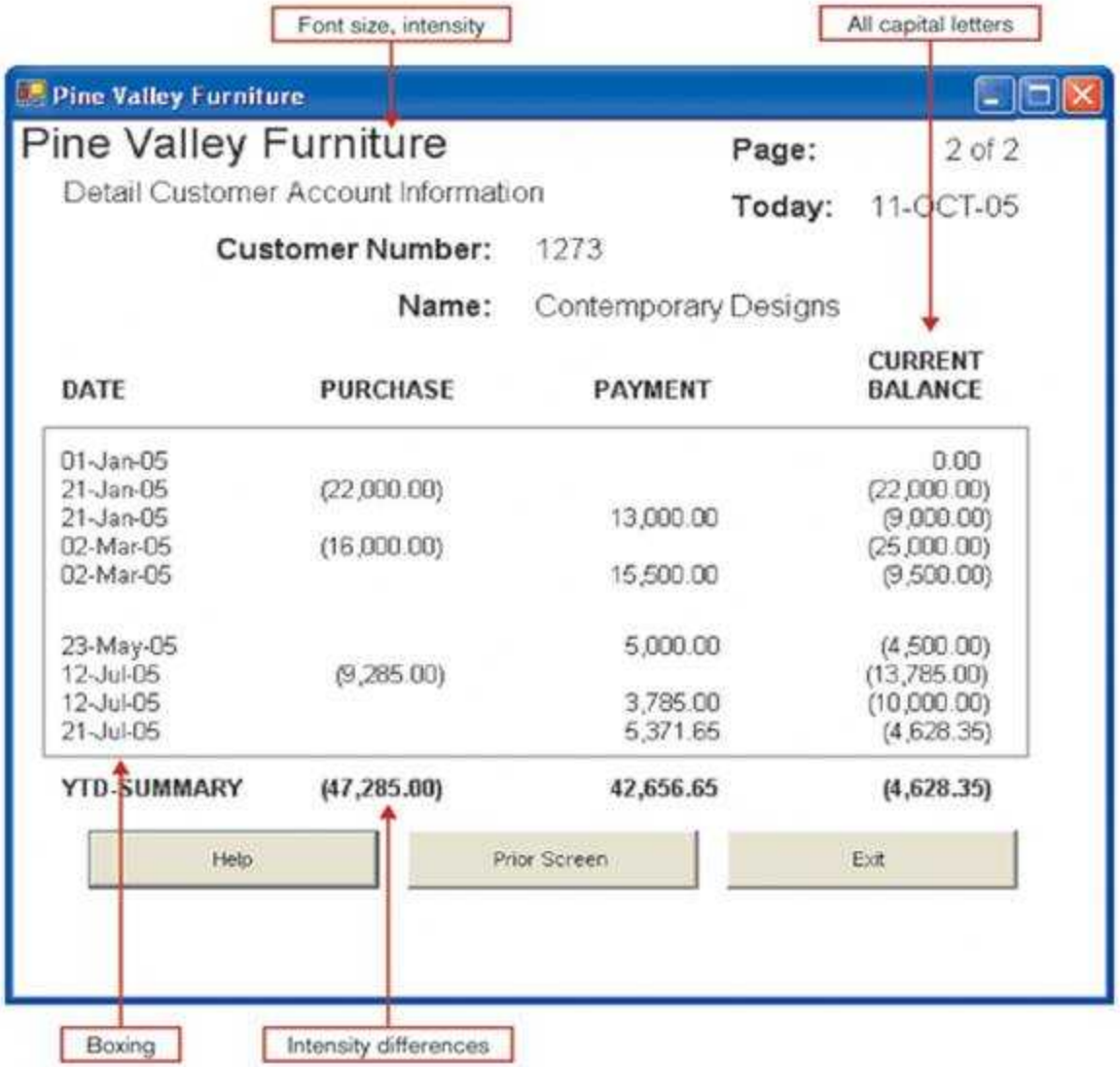
Uses of Highlighting in Forms and Reports

- Notify users of errors in data entry or processing.
- Provide warnings regarding possible problems.
- Draw attention to keywords, commands, high-priority messages

Methods for Highlighting

- Blinking
- Audible tones
- Intensity differences
- Size differences
- Font differences
- Boxing
- Underlining
- All capital letters

Figure 11-6 Customer account status display using various highlighting techniques (Pine Valley Furniture)

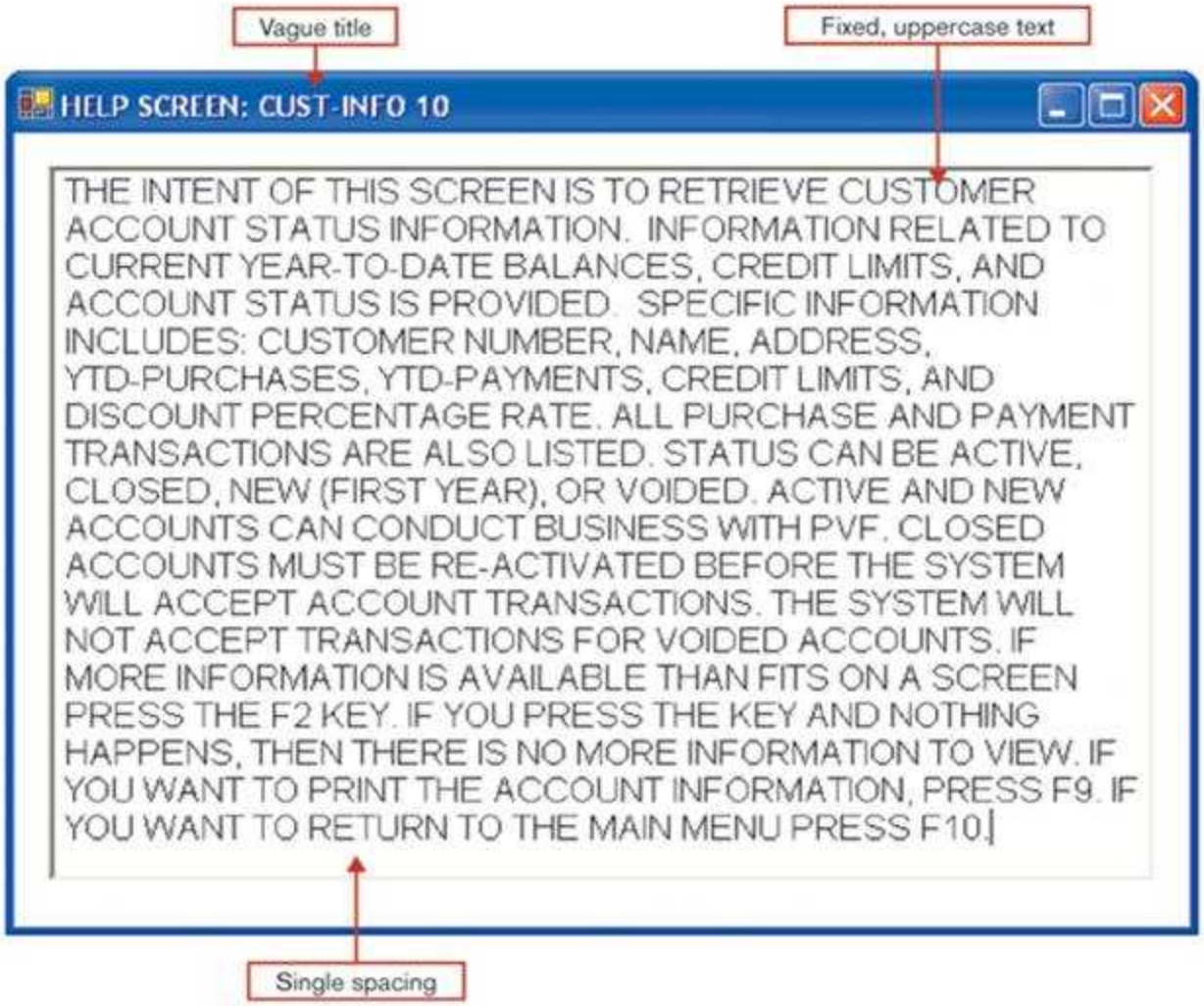


Highlighting can include use of upper case, font size differences, bold, italics, underline, boxing, and other approaches.

Guidelines for Displaying Text

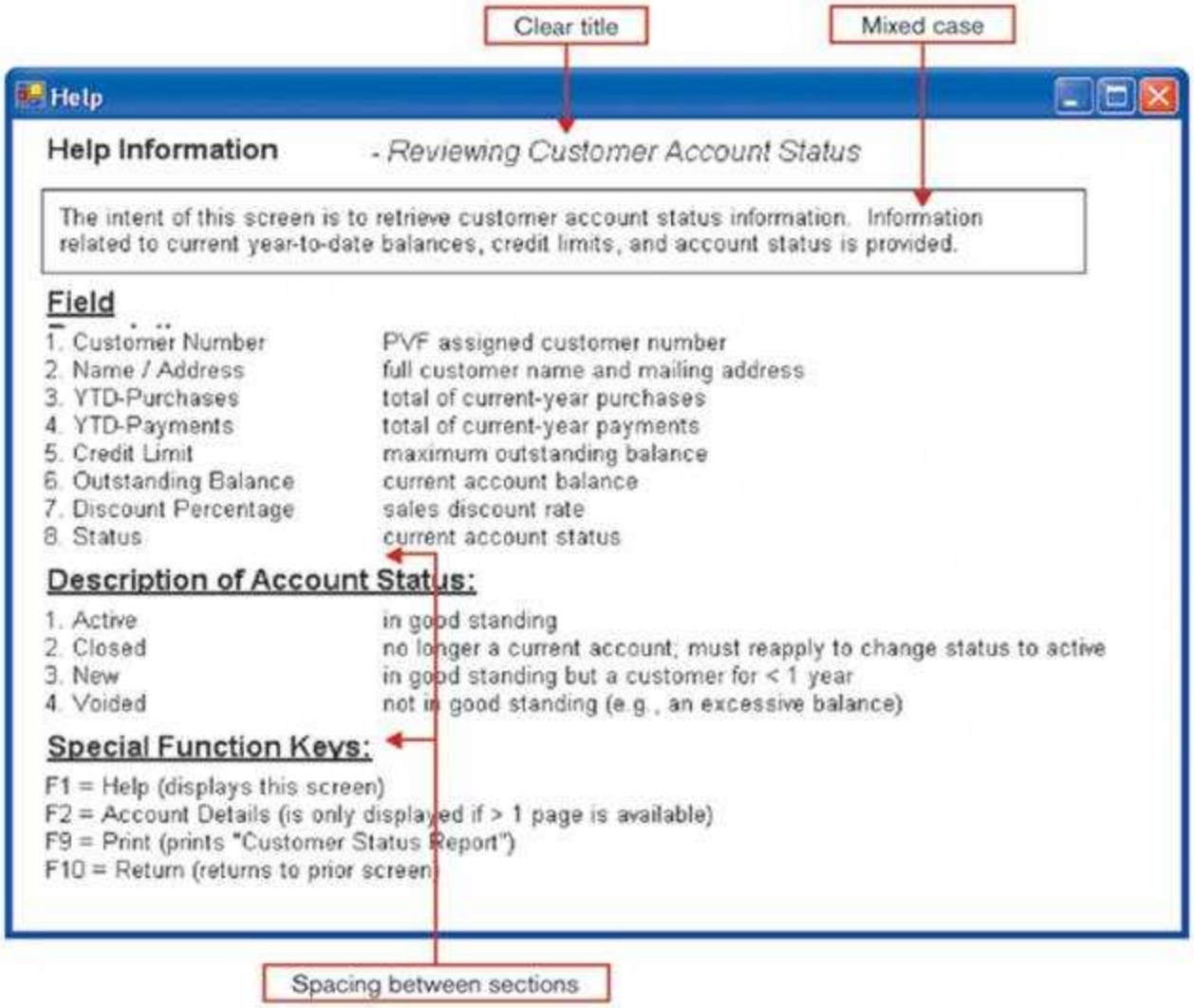
- Case: mixed upper and lower case, use conventional punctuation
- Spacing: double spacing if possible, otherwise blank lines between paragraphs
- Justification: left justify text
- Hyphenation: no hyphenated words between lines
- Abbreviations: only when widely understood and significantly shorter than full text

Figure 11-7a Contrasting the display of textual help information - Poorly designed help screen with many violations of the general guidelines for displaying text



A poor help screen design

Figure 11-7b Contrasting the display of textual help information - An improved design for a help screen



A better help screen design

Guidelines for Tables and Lists

- Labels
  - All columns and rows should have meaningful labels.
  - Labels should be separated from other information by using highlighting.
  - Redisplay labels when the data extend beyond a single screen or page.

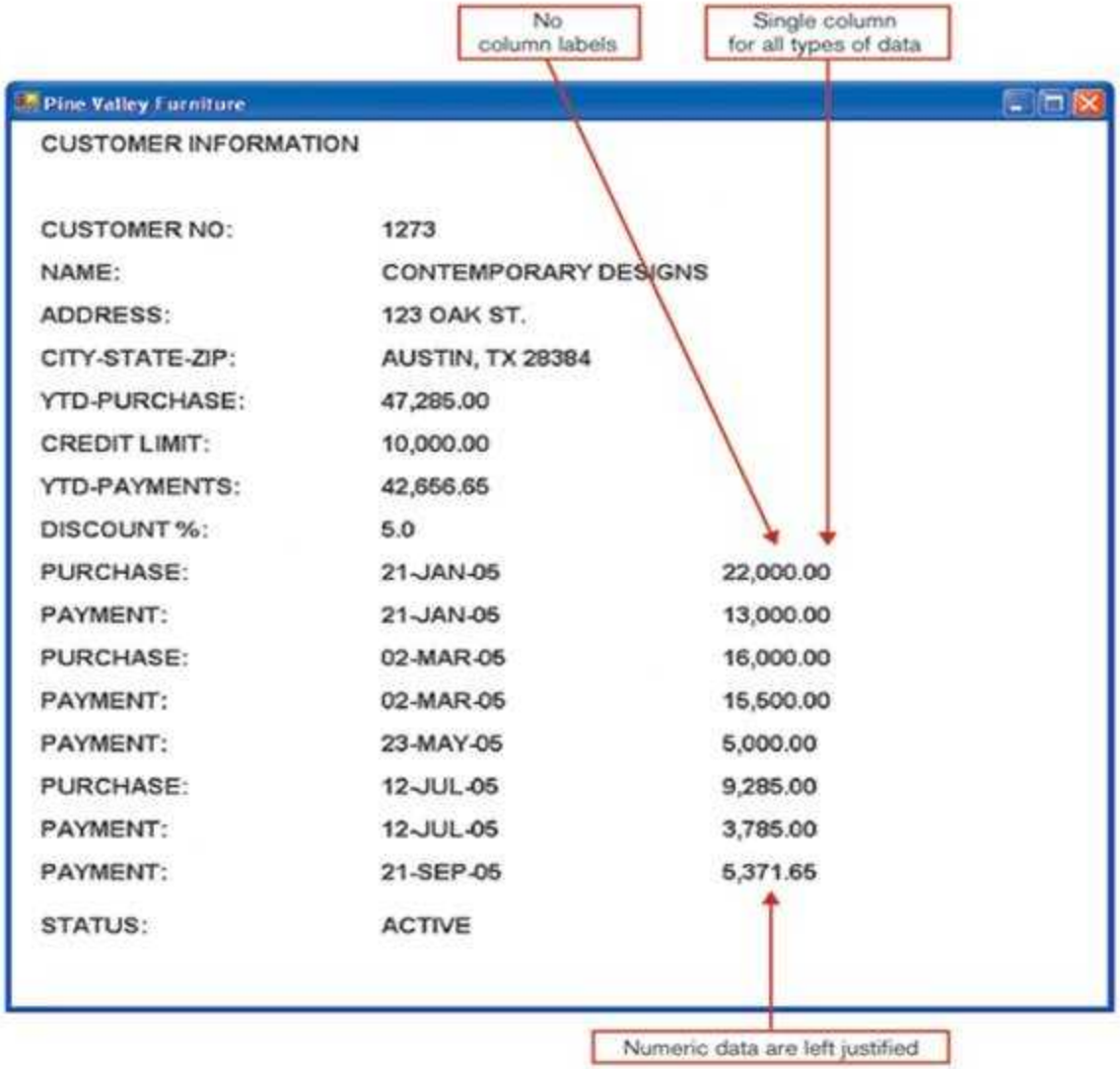
Guidelines for Tables and Lists (cont.)

- Formatting columns, rows and text:
  - Sort in a meaningful order.
  - Similar information displayed in multiple columns should be sorted vertically.
  - Columns should have at least two spaces between them.
  - Allow white space on printed reports for user to write notes.
  - Use same family of typefaces within and across displays and reports.
  - Avoid overly fancy fonts.

Guidelines for Tables and Lists (cont.)

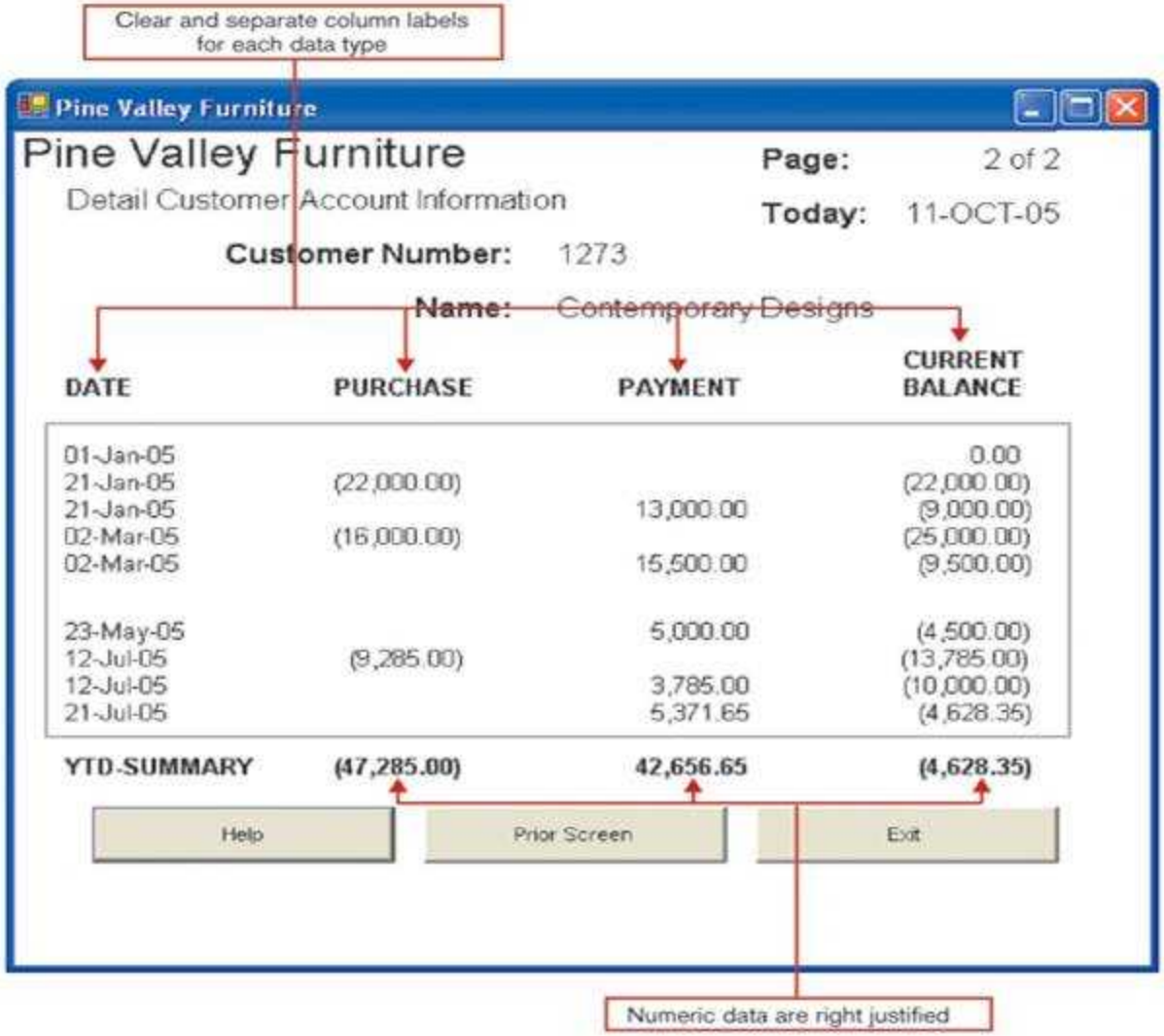
- Formatting numeric, textual and alphanumeric data:
  - Right justify numeric data and align columns by decimal points or other delimiter.
  - Left justify textual data. Use short line length, usually 30 to 40 characters per line.

Figure 11-8a Contrasting the display of tables and lists (Pine Valley Furniture) - Poorly designed form



A poor table design

Figure 11-8b Contrasting the display of tables and lists (Pine Valley Furniture) - Improved design for form



A better table design

Interaction Methods

- **Methods of Interacting**
  - Command Language Interaction
    - Users enter explicit statements into a system to invoke operations. This type of interaction requires users to remember command syntax and semantics.
  - Menu Interaction
    - A list of system options is provided
    - A specific command is invoked by user selection of a menu option
    - Menu complexity varies according to needs of system and capabilities of development environment
    - Two common placement methods
      - Pop-up
      - Drop-down

Interaction Methods

- Form Interaction
  - **Allows users to fill in the blanks when working with a system**

Figure 11-3
A data input screen designed in Microsoft's Visual Basic .NET

Interaction Methods

- Object-Based Interaction
  - Symbols are used to represent commands or functions
  - Icons
    - **Graphic symbols that look like the processing option they are meant to represent**
    - **Use little screen space**
    - **Can be easily understood by users**

Interaction Methods

- Natural Language Interaction
  - Inputs to and outputs from system are in a conventional speaking language like English

Introduction to HCI

What is human-computer interaction (HCI)?

- HCI is the study and the practice of usability.
- It is about understanding and creating software and other technology that people will want to use, will be able to use, and will find effective when used.
- HCI is the study of how people use computer systems to perform certain tasks.
- HCI tries to provide us with all understanding of the computer and the person using it, so as to make the interaction between them more effective and more enjoyable.

HCI

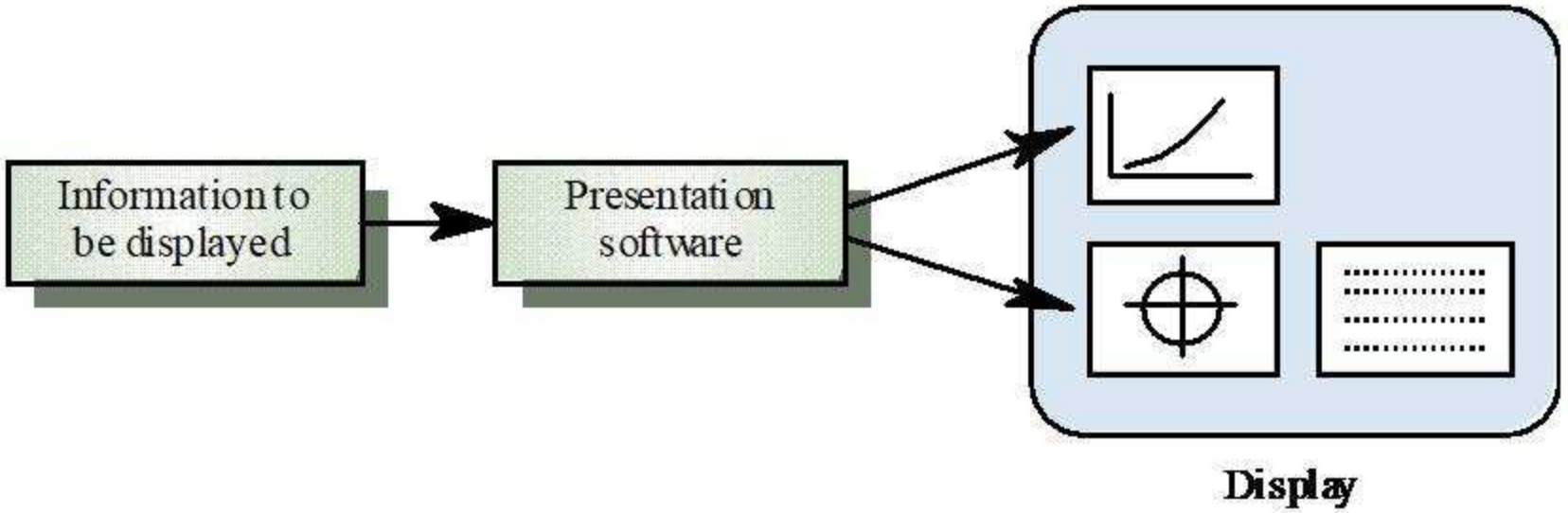
- Humans interact with computers through a user interface. This includes software, such as what is displayed on the computer monitor, and hardware, such as the mouse, keyboard and other peripheral devices. As a result, the study of HCI focuses on user satisfaction. Attention to human machine interaction is important, because a poor interface can make it hard for users to benefit from even the simplest systems.
- Usability and user experience goal awareness is essential to all HCI design, as follows:
    - Usability: Central to interaction design and operations through specific computer system criteria, including efficiency, safety, utility and learning/retention.
    - User Experience: Focuses on creating systems that are satisfying, enjoyable, entertaining, helpful, motivating, aesthetically pleasing, creativity supportive, rewarding, fun and emotionally fulfilling

HCI

- **Another consideration in studying or designing HCI is that user interface technology changes rapidly, offering new interaction possibilities to which previous research findings may not apply. Finally, user preferences change as they gradually master new interfaces.**

Information presentation

- Information presentation is concerned with presenting system information to system users
- The information may be presented directly or may be transformed in some way for presentation
- The Model-View-Controller approach is a way of supporting multiple presentations of data



Information display factors

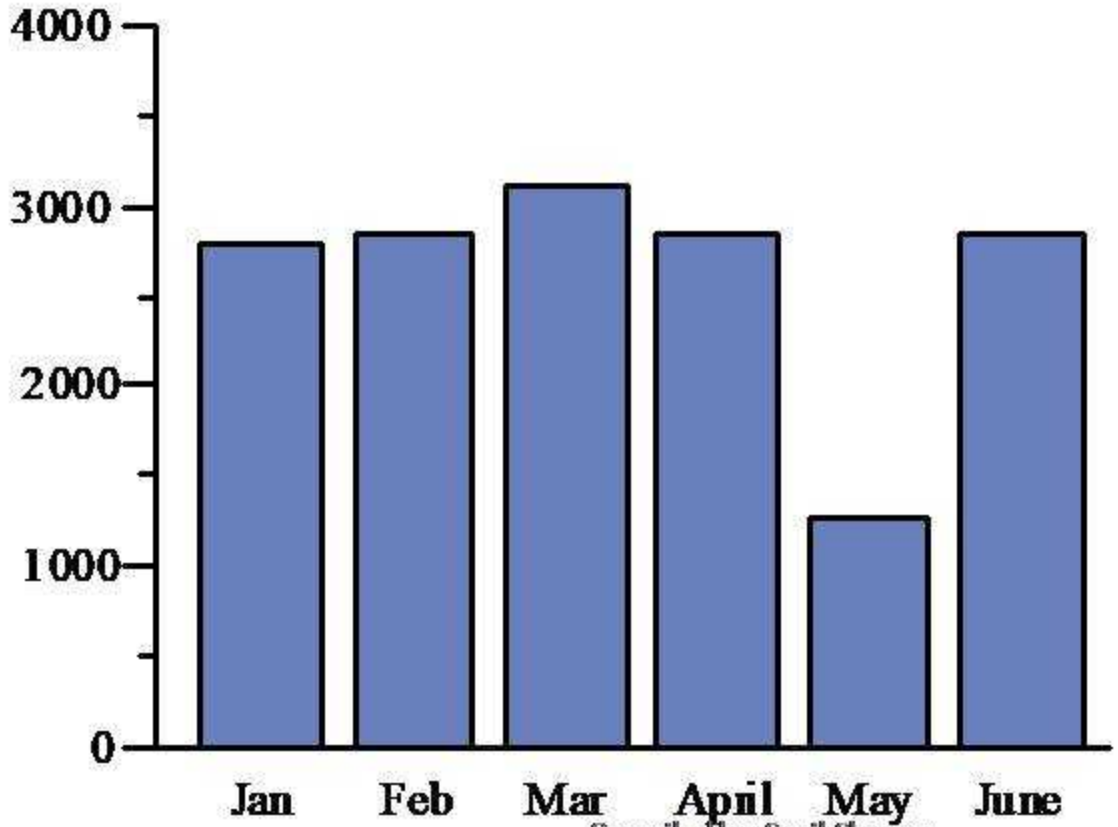
- Is the user interested in precise information or data relationships?
- How quickly do information values change? Must the user take some action in response to a change?
- Is the information textual or numeric? Are relative values important?

Analog vs. digital presentation

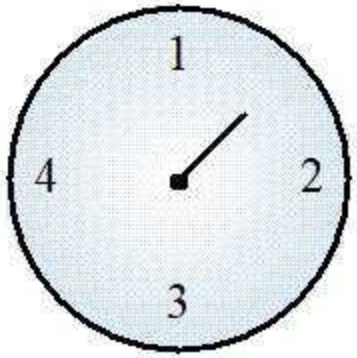
- Digital presentation
  - Compact - takes up little screen space
  - Precise values can be communicated
- Analog presentation
  - Easier to get an 'at a glance' impression of a value
  - Possible to show relative values
  - Easier to see exceptional data values

Alternative information presentations

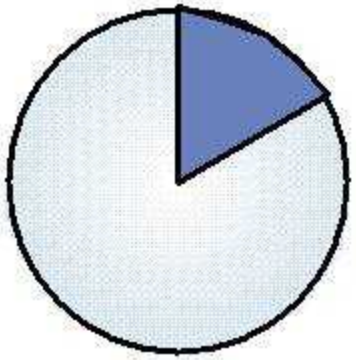
Jan	Feb	Mar	April	May	June
2842	2851	3164	2789	1273	2835



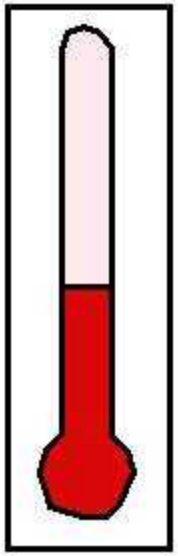
Information display



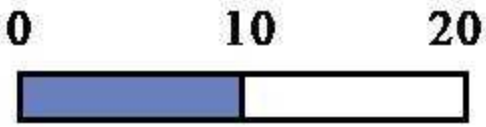
Dial with needle



Pie chart

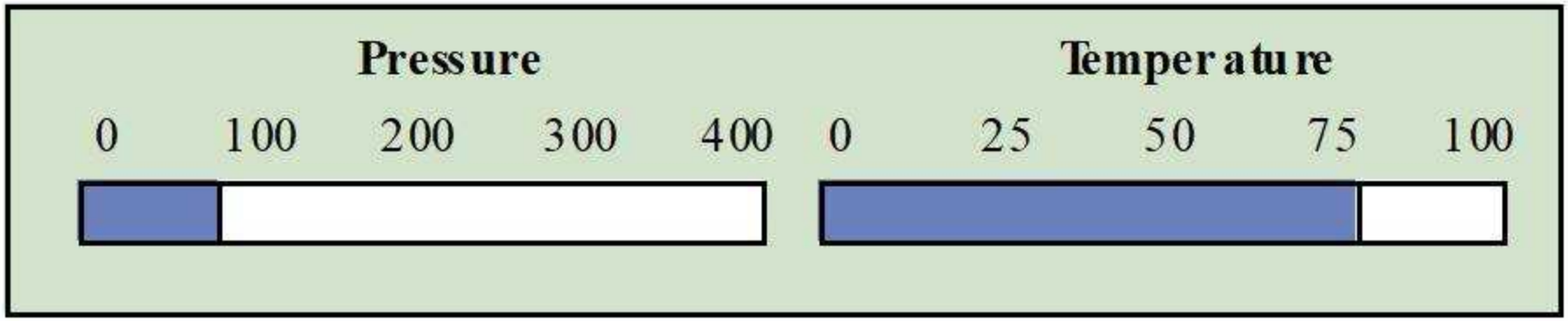


Thermometer



Horizontal bar

Displaying relative values



Tables vs. Graphs

- Use tables for reading individual data values
- Use graphs for:
  - Providing quick summary
  - Displaying trends over time
  - Comparing points and patterns of variables
  - Forecasting activity
  - Simple reporting of vast quantities of information

Figure 11-9 Tabular report illustrating numerous design guidelines (Pine Valley Furniture)

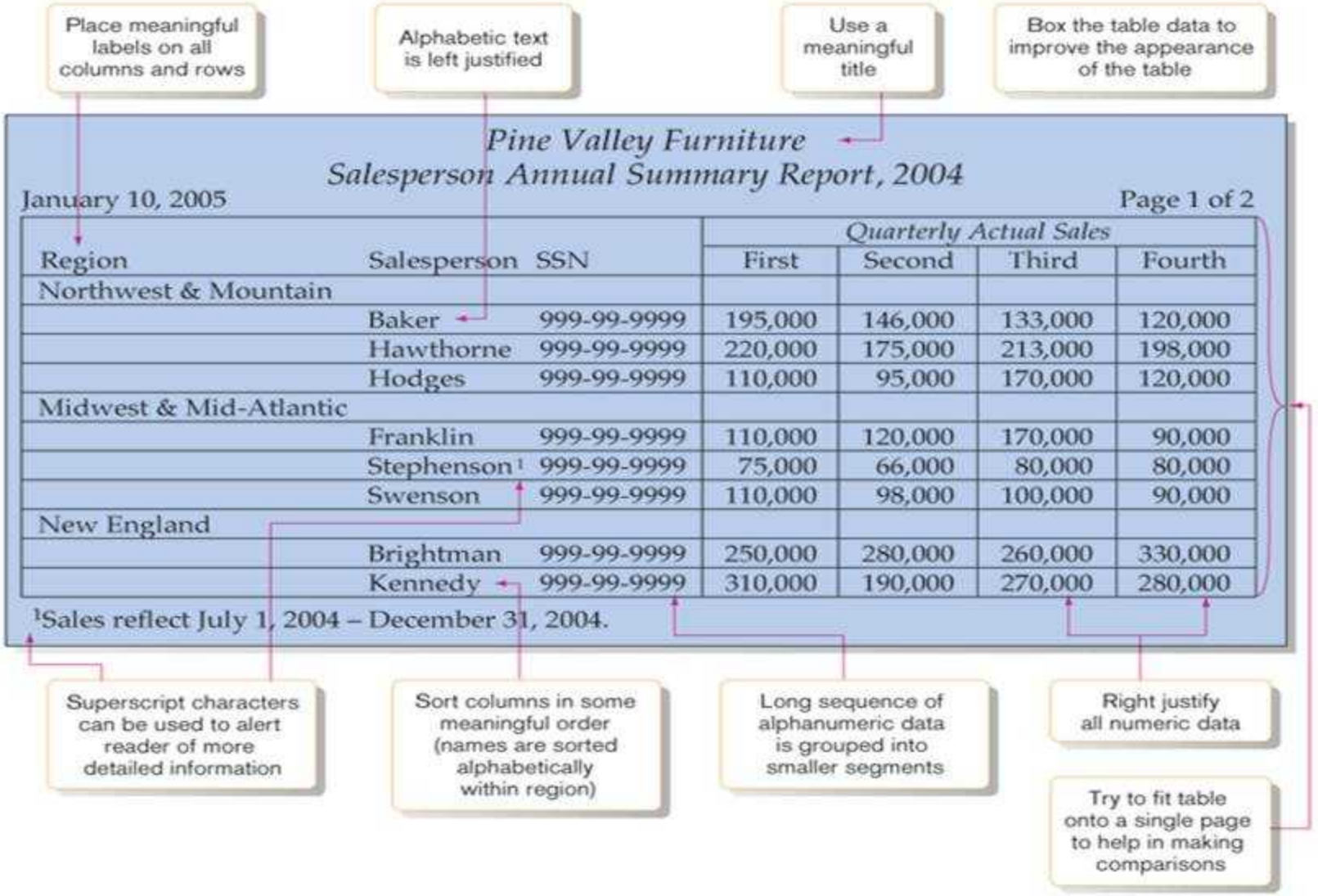
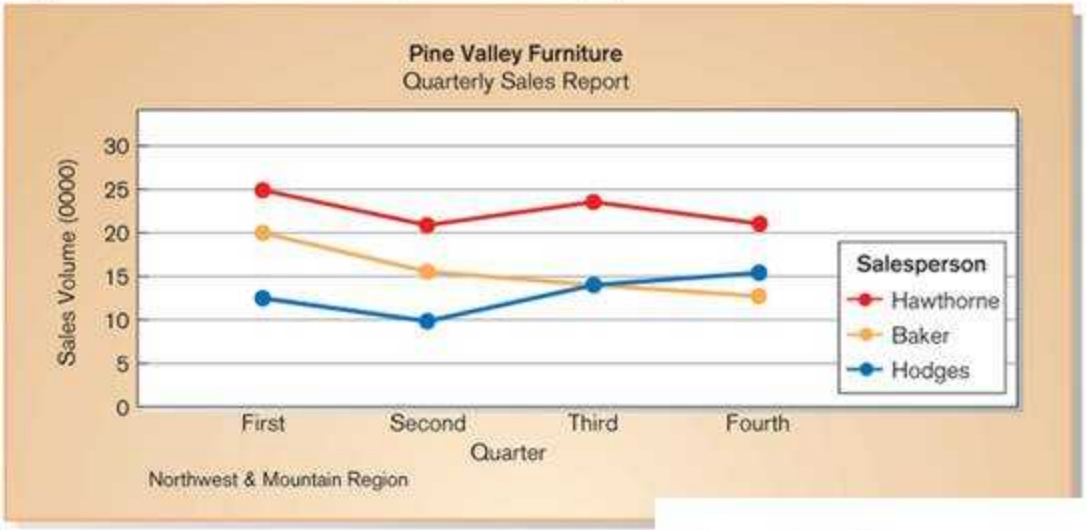
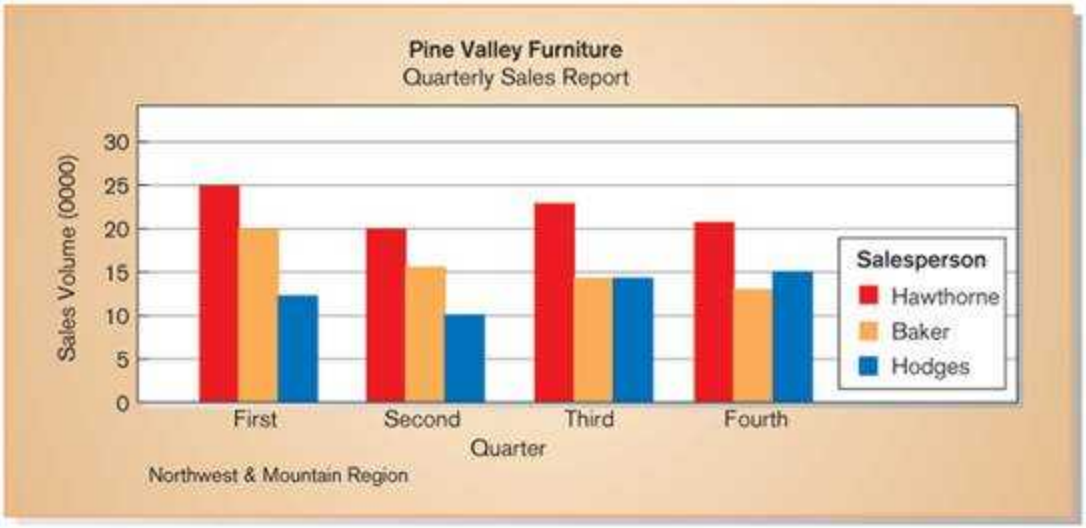


Figure 11-10a Graphs for comparison - Line graph



Bar and line graphs give pictorial summary information that can enhance reports and forms.

Figure 11-10b Graphs for comparison - Bar graph



User interface evaluation

- Some evaluation of a user interface design should be carried out to assess its suitability
- Full scale evaluation is very expensive and impractical for most systems
- Ideally, an interface should be evaluated against a usability specification
- However, it is rare for such specifications to be produced

Usability attributes

Attribute	Description
Learnability	How long does it take a new user to become productive with the system?
Speed of operation	How well does the system response match the user's work practice?
Robustness	How tolerant is the system of user error?
Recoverability	How good is the system at recovering from user errors?
Adaptability	How closely is the system tied to a single model of work?

Simple evaluation techniques

- Questionnaires for user feedback
- Video recording of system use and subsequent evaluation.
- collect information about facility use and user errors.
- The provision of a “gripe” button for on-line user feedback.