We enable you to leverage knowledge anytime, anywhere!

Education and Research

Overview of Software Design

Education & Research

"© 2009 Infosys Technologies Ltd. This document contains valuable confidential and proprietary information of Infosys. Such confidential and proprietary information includes, amongst others, proprietary intellectual property which can be legally protected and commercialized. Such information is furnished herein for training purposes only. Except with the express prior written permission of Infosys, this document and the information contained herein may not be published, disclosed, or used for any other purpose."

Confidential Information

- *This Document is confidential to Infosys Technologies Limited. This document contains information and data that Infosys considers confidential and proprietary ("Confidential Information").*
- *Confidential Information includes, but is not limited to, the following:*
  - *Corporate and Infrastructure information about Infosys;*
  - *Infosys' project management and quality processes;*
  - *Project experiences provided included as illustrative case studies.*
- *Any disclosure of Confidential Information to, or use of it by a third party, will be damaging to Infosys.*
- *Ownership of all Infosys Confidential Information, no matter in what media it resides, remains with Infosys.*
- *Confidential information in this document shall not be disclosed, duplicated or used – in whole or in part – for any purpose without specific written permission of an authorized representative of Infosys.*

Course Objectives

- To understand the principles of software design
- To know the steps involved in design process
- To go through key issues in software design
- To discuss various design notations
- To get an overview of various design tools

© 2009 Infosys Technologies Ltd.

Agenda

- Introduction to Software Design
- Software Design Concepts
- Software Design Considerations
- Software Design Process
  - High-level Design
  - Detailed Design
- Software Design Strategies and Methods
- Software Design Issues
- Software Design Notations
- Software Design Tools

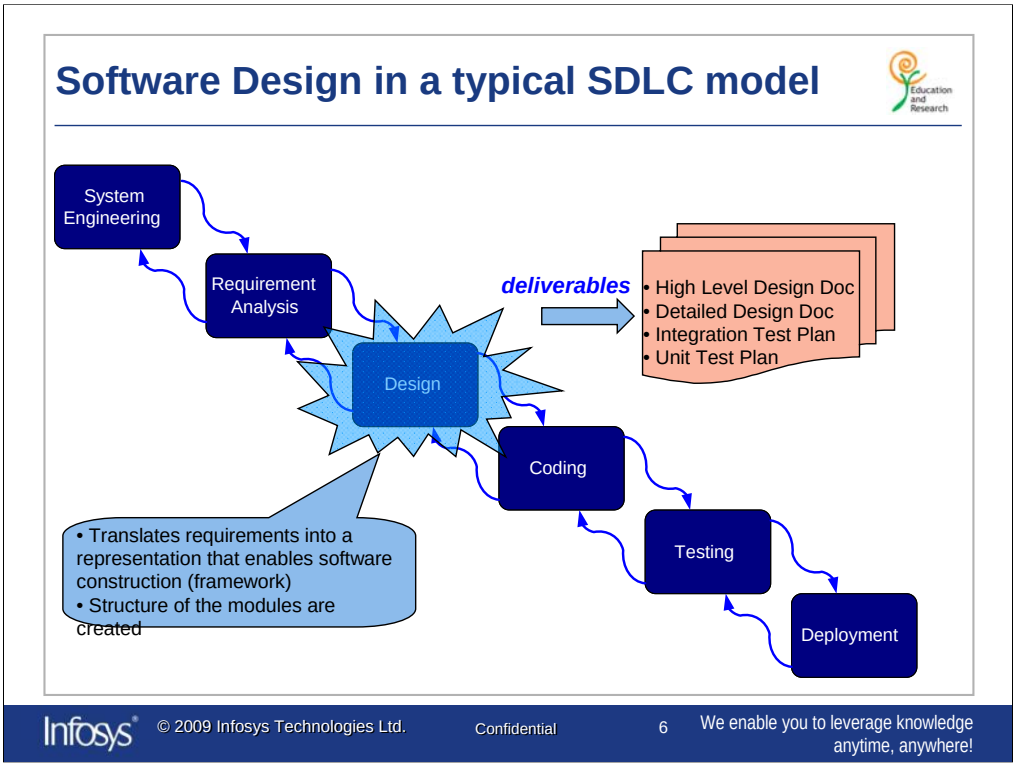
What is Software Design?

Design is both "the process of defining the architecture, components, interfaces, and other characteristics of a system or component" and "the result of [that] process."
- IEEE definition [IEEE 610.12-90]

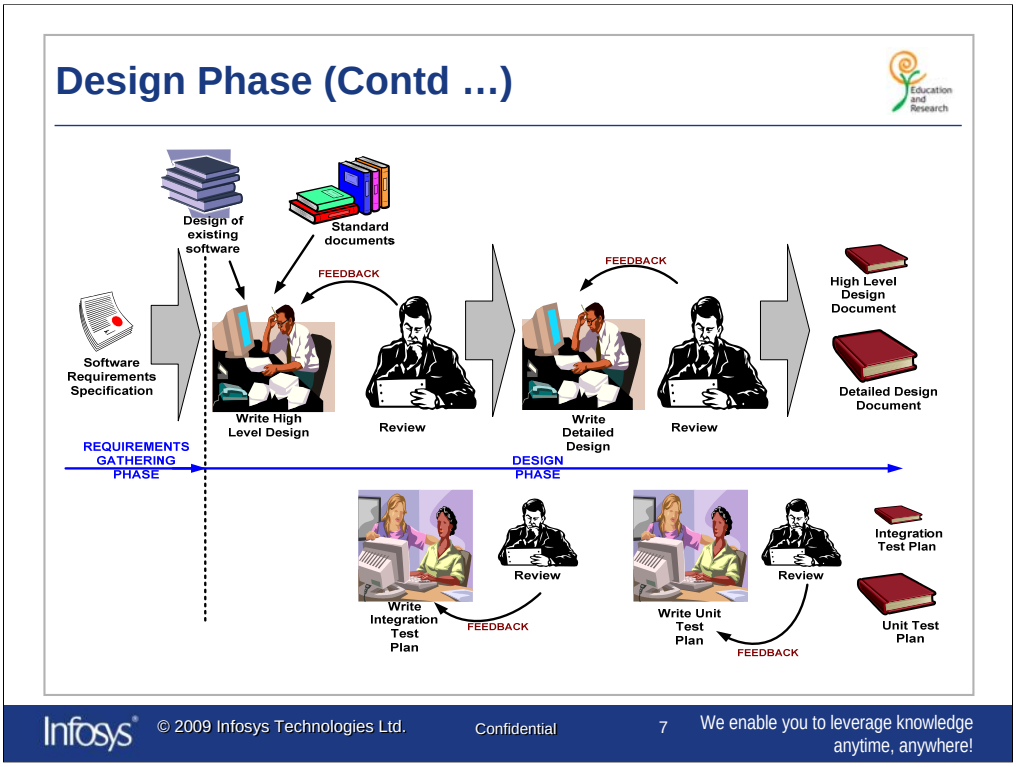
Software Design is a process of problem-solving and planning for a software solution which includes low-level component and algorithm implementation issues as well as the architectural view

Software Design describes how software is decomposed and organized into components—and the interfaces between those components. It also describes the components at a level of detail that enable their construction.

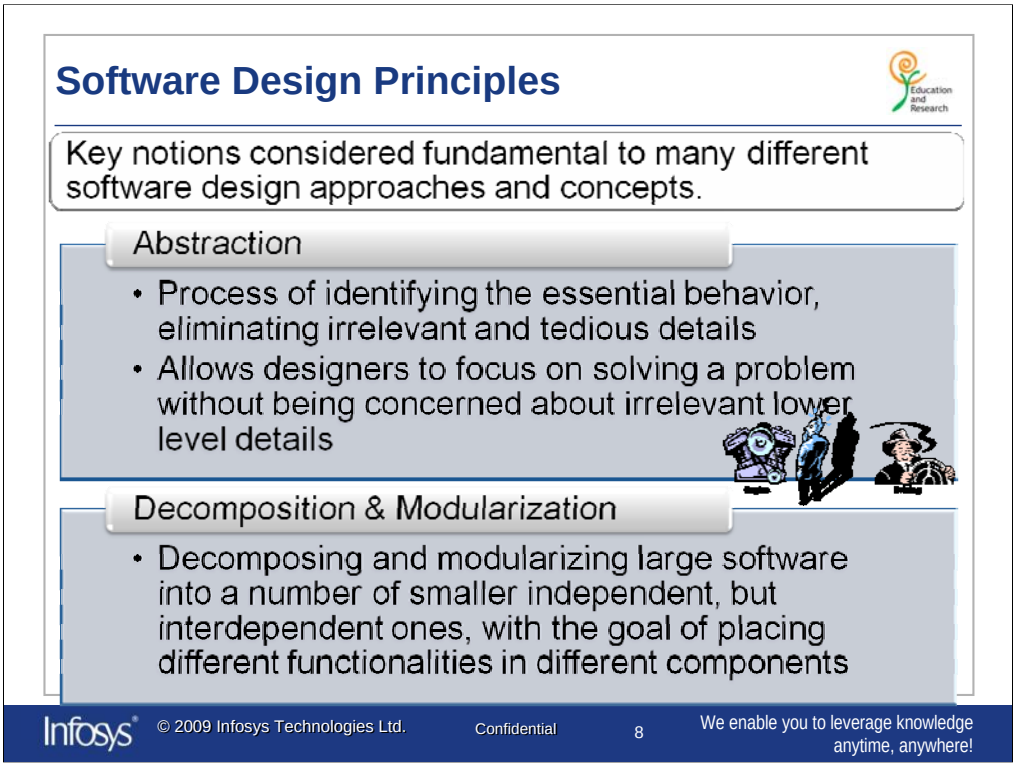
© 2009 Infosys Technologies Ltd.



Software design is the activity in Software development life cycle in which software requirements are analyzed to create the description of the internal structure of the software. This serves as the basis for its construction.

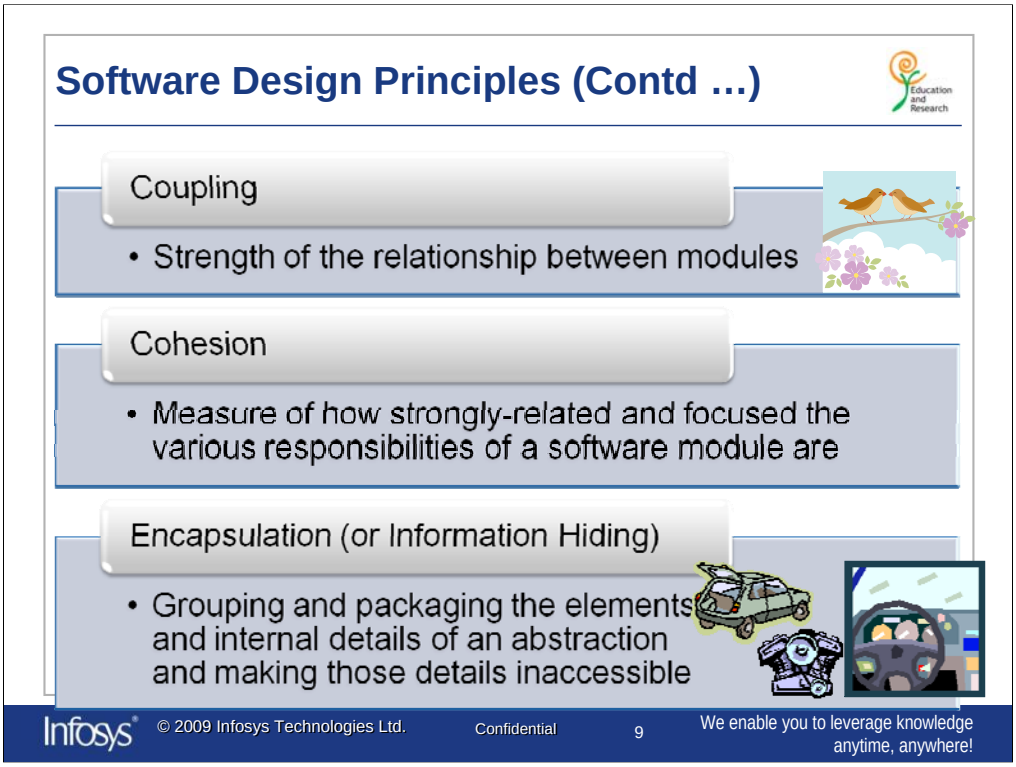


Software design phase takes requirements as input and translates it into representation that enables software construction. The output of this phase includes high level design document, detailed design document, unit test plan and integration test plan.



Abstraction: Representing the essential characteristics of an object that distinguish it from all other types of objects and hence provide crisply defined conceptual boundaries, from the perspective of the viewer/user. Focus here is on the observable behavior of an object. Classical example for this is a 'route map'. Maps of the world, a country, state or a city show different things like country / state boarders, rivers, mountains, roads, etc. One needs to buy a right map based on his/her requirement (to see different countries, cities of a state or road details of a city). Another example: A person buying a car will look mostly outside parts of the car than a mechanic who is more interested on the inner working of the car.

Modularity: The property of a system that has been decomposed into a set of cohesive and loosely coupled modules. Another way to manage complexity of systems by breaking them into better manageable modules. Modularity is the property of a system that has been decomposed into a set of cohesive but loosely coupled modules.



Coupling is also referred as dependency, It is the degree to which each module in the program

depends on each one of the other modules. Low coupling among program modules in a system is recommended for good design.

Disadvantages of Tightly coupled systems

- Change in one module results in changes in other modules
- It is difficult to reuse a particular module as the dependent modules also need to be included

Advantages of Loosely coupled systems

- Change in one module does not cause changes in other modules
- It is easier to reuse a particular module as there are no dependencies

Cohesion: It is the measure of how strongly-related the responsibilities of a single software module are.

Modules with high cohesion are preferred because high cohesion is associated with characteristics such as robustness, reliability, reusability, and understandability. Low cohesion results in difficulty in maintenance/testing/reuse/readability.

Coupling is normally contrasted with cohesion. High coupling mostly correlates with low cohesion and vice versa.

<u>Encapsulation (Information Hiding)</u>: Means '*in a capsule'*. Abstraction is achieved mainly through information hiding. Objects encapsulate data and implementation details. To the outside world, an object is a black box that exhibits a certain behavior. An object exposes its behavior by means of methods or functions. E.g. The dashboard of a car acts as an interface to the external world (hiding its inner details).

Software Design Principles (Contd …)

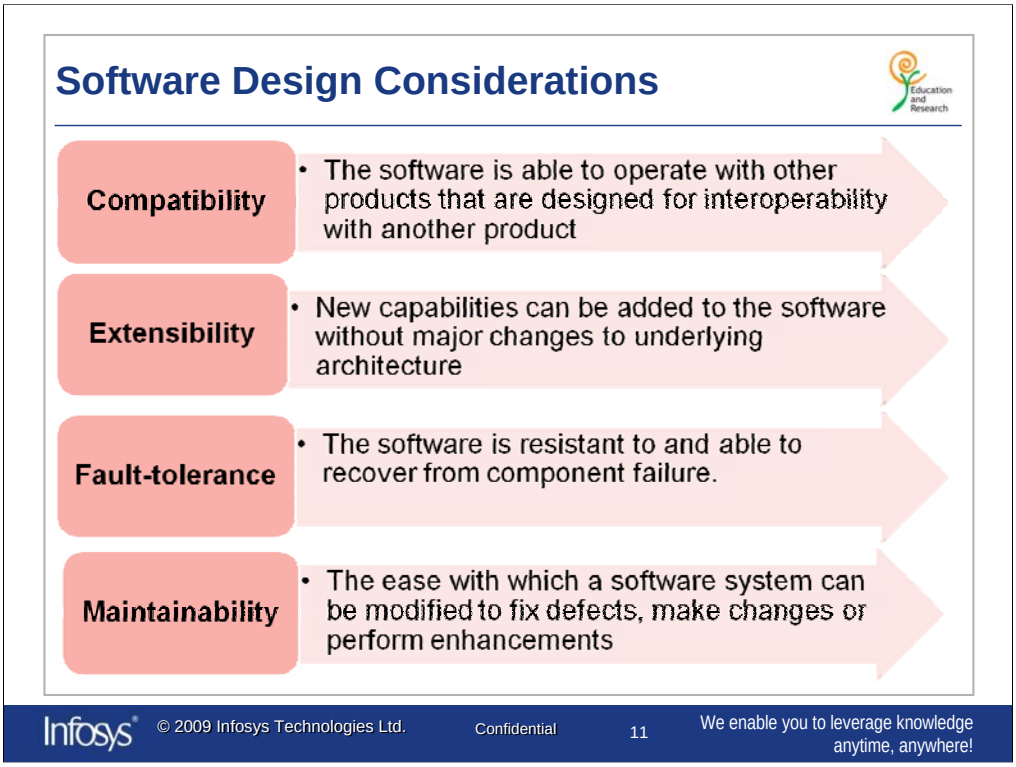
Separation of interface and implementation

- Defining a component by specifying a public interface, known to the clients, separate from the details of how the component is realized.(E.g. TV Remote interface)

Sufficiency, completeness and primitiveness

- Ensuring that a software component captures all the important characteristics of an abstraction, and nothing more
- To ensure that software does whatever is asked for, in all respects and in a very simple way as possible

© 2009 Infosys Technologies Ltd.



Software compatibility can refer to the ability for the software to run on a particular OS or environment or hardware infrastructure like CPU architecture etc.

A forward-compatible system is expected to handle input which is intended for a newer version, by ignoring the one that are not known and selecting the known set of data that the system is capable of handling.

A product is said to be backward compatible when it is able to fully take the place of an older product, by inter-working with products that were designed for the earlier release of the product. e.g. Software that works on older versions of an operating system is said to be backwards compatible.

Extensibility: Being one of the principles of software design, extensibility considers the future growth of software in terms of adding new functionality or modifying existing functionality. It is the ability of a software to extend a system.

Fault- tolerance: Software's ability to detect and recover from a fault either in hardware or software in which the application is running. Constructing highly available and reliable computing systems requires this necessary 'fault – tolerance' characteristics.

Software Design Considerations

Modularity	• The resulting software comprises well defined, independent components, that lead to better maintainability
Reliability	• The software is able to perform a required function under stated conditions for a specified period of time
Reusability	• The likelihood a segment of source code that can be used again to add new functionalities with slight or no modification
Robustness	• The ability to operate under stress or tolerate unpredictable or invalid input

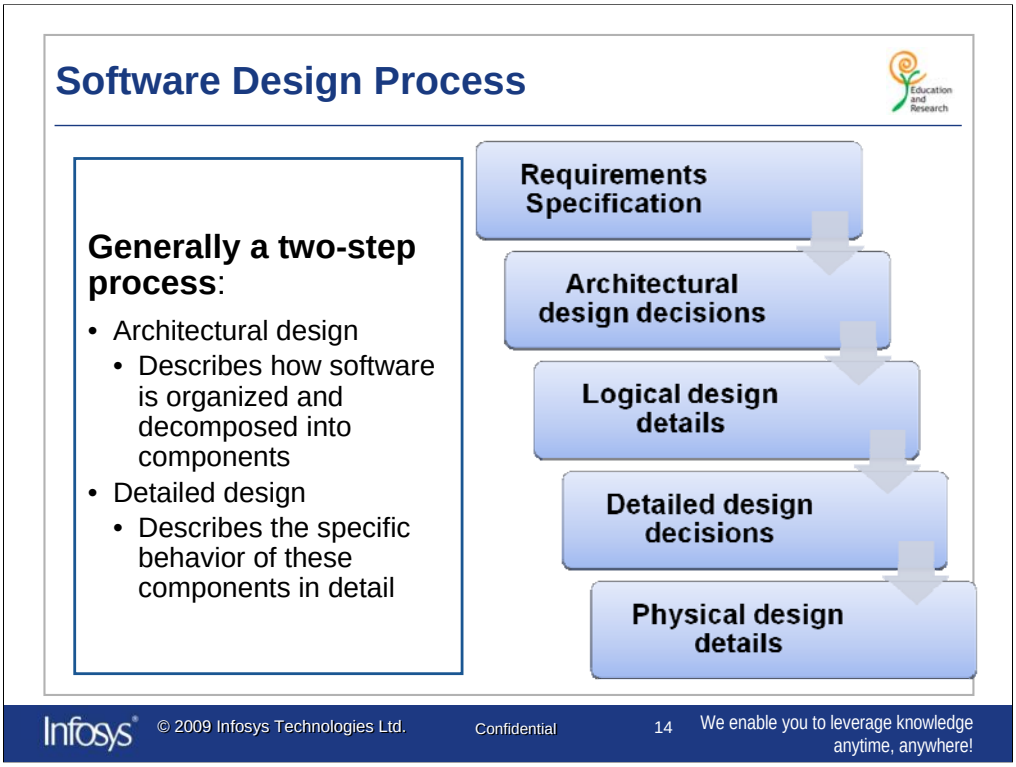
© 2009 Infosys Technologies Ltd.

Software Design Considerations

Security	• To protect information and information systems from unauthorized access, use, disclosure, disruption, modification or destruction
Usability	• The elegance and clarity with which the interaction with a software system is designed. The user interface must be intuitive (and often aesthetically pleasing) to its target user/audience.
Interoperability	• The ability of two or more systems or components to exchange information and to use the information that has been exchanged

Infosys © 2009 Infosys Technologies Ltd. Confidential

We enable you to leverage knowledge anytime, anywhere!



Software design process - It is generally a two step process.

Architectural design - This describes how a program or software is decomposed and organized into components.

Detailed design - This describes the specific behavior of these components. The output of this process which are set of models and artifacts documents the major design decisions that are taken. This gives details that are necessary for construction.

Detailed Design Decision: The following points are looked at during detailed design

Classification - The kind of component, such as a subsystem, module, class, package, function, file, etc.

Definition - The specific purpose and semantic meaning of the component.

Responsibilities - The primary responsibilities and/or behavior of this component.

Constraints - Any relevant assumptions, limitations, or constraints for this component.

Composition - A description of the use and meaning of the subcomponents that are a part of this component.

Uses/Interactions- A description of this components collaborations with other components.

Resources- A description of any and all resources that are managed, affected, or needed by this entity.

Processing - A description of precisely how this components goes about performing the duties necessary to fulfill its responsibilities.

Interface/Exports- The set of services (resources, data, types, constants, subroutines, and exceptions) that are provided by this component.

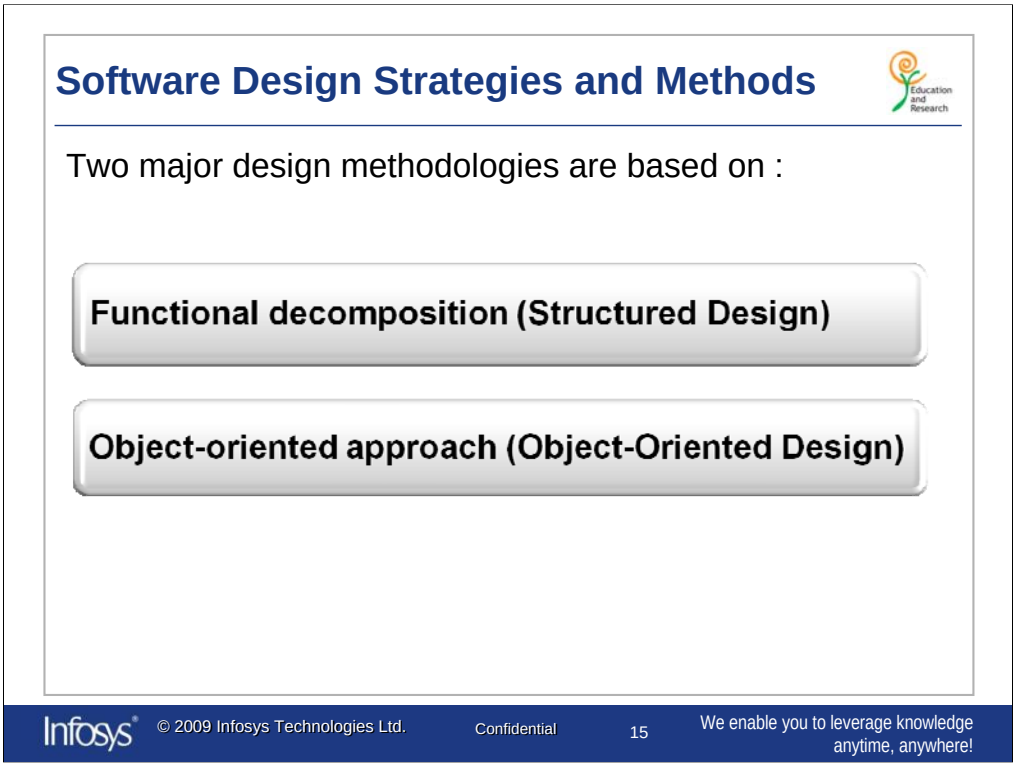
Logical and Physical Design:

The logical design is more conceptual and abstract than the physical design.

In the *logical design***,** the logical relationships among the objects is given importance.

In the *physical design*, The most effective way of storing and retrieving the objects is concentrated upon.

The physical implementation of the logical model may require some changes due to the system parameters such as size of machine, number of users, storage capacity, type of network, and software.



There exist various strategies or methods to help guide the design process. Such methods are useful for knowledge transfer and as a common framework for software engineering teams. Two major design methodologies are based on :

- Functional Decomposition or Structured Design
- Object-oriented approach or OO Design

Functional Decomposition or Structured Design

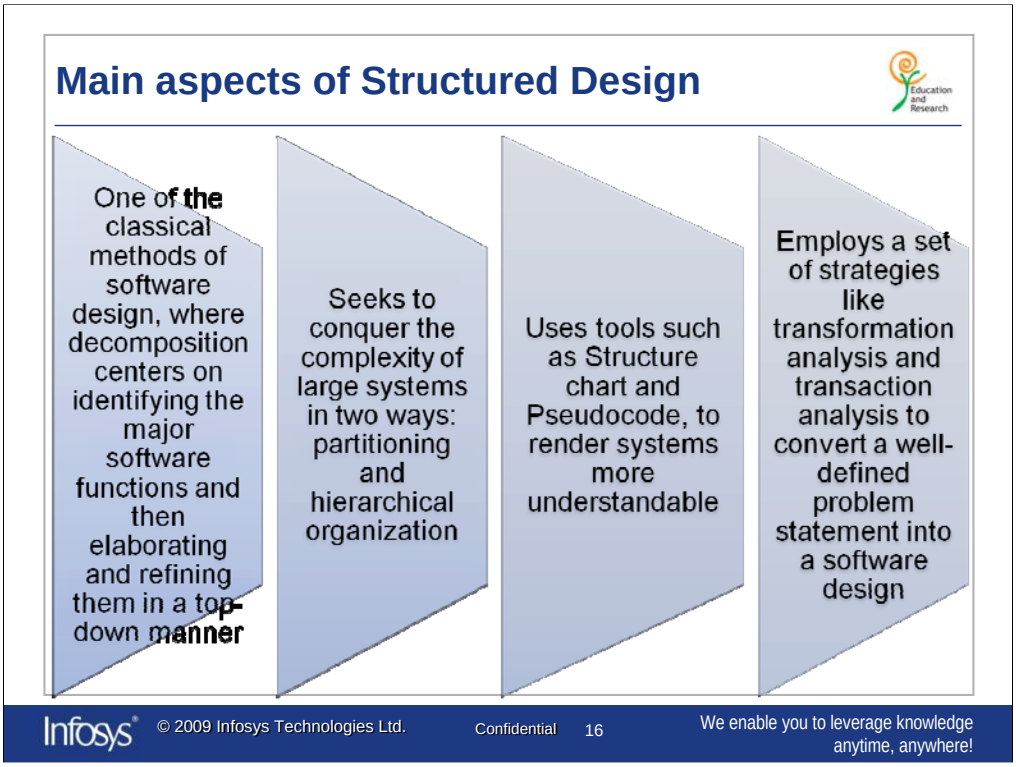
Structured design is a combination of both data model and process model.

Programs were constituted with algorithms in form of procedures or functions and data structures to handle the data. A main concern in structured system design is that there are no association between the Functionality and data. If there is a change in the data, we have to change in the each and every function where the data is referred. *As system grows and changes, such models are very hard to maintain.*

OO Design:

To overcome the data/function divide split, Object oriented model came in to existence where bridges this gap through an object. This is an approach for organizing software by combining data and methods into an object. Models the system as “real world” objects in a way that is very close to the actual domain terminology.

Main building block in OO: Concept of object overcomes the data function divide where it integrates the data in form of attributes and function as methods into objects.



Structured Design (SD):

•It is a data flow-oriented design approach

•It is one of the most popular methodology of software design

•It is easy to use and there is an evaluation criteria that can serve as guide in the software design

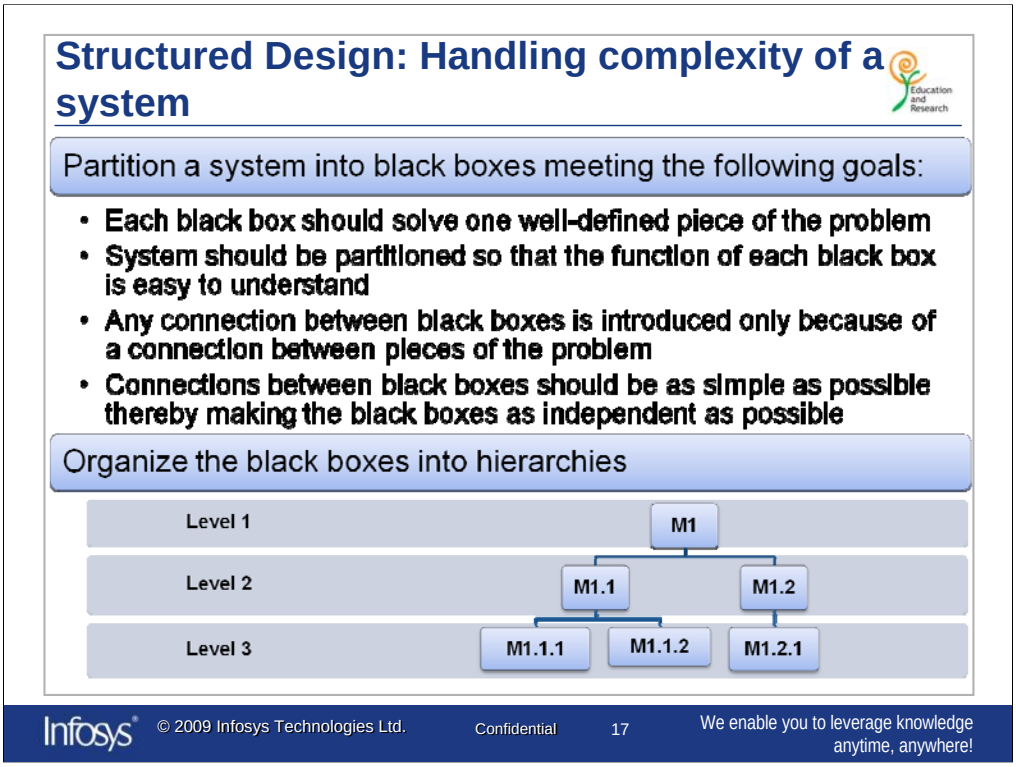
•The main notational scheme that Structured Design uses is the data flow diagram (DFD)

SD views systems from two perspectives :

•As the flow of data

•The transformations that data flow undergoes through a system.

Since data flows and transform actions are the only characteristics depicted in the DFDs, the element of time is not present. Thus, the designer can just focus on the transformations of the data flows through a system. Through the perception of the system as data flows and transforms, there is minimal variation in the construction of the system model, as a result the shape or structure of the system is maintained.



The SD process suggests a differentiation between the logical design (abstract) and the physical design (implementation).

Through the analysis of the data flows and the transformations, the designer can derive a logical solution devoid of implementation considerations. This early logical solution will no t have details; will not be precise; and cannot be implemented immediately.

Once this logical solution is able to satisfy the requirements or meet the objectives, the designer will then make the necessary changes so that the solution can be implemented. In other words, the logical design (abstract) needs to be converted to the physical design (implementation).

Tools in Structured Design

- Structured design uses tools such as *structure charts* and *pseudo code*
- Output from structured analysis such as data flow diagrams serves as input to structure chart
- Major tool used in structured design to depict the structure of a system is the structure chart.

Infosys © 2009 Infosys Technologies Ltd. Confidential 18 We enable you to leverage knowledge anytime, anywhere!

The basic approach of SD is to start off with a system specification that identifies the inputs, the desired outputs and a description of the functional aspects of the systems that transforms the data. This specification is used for the graphic representation of the system, the data flow diagram.

The DFDs are derived from the existing actual processes in the system which are iteratively refined and decomposed to a final logical solution.

- The DFDs should be checked for consistency and data conservation through each iteration.
- A data dictionary should then be built in a logical form specifying the content of the data, the data flows and all the data forms in the DFDs.
- The data dictionary must be well correlated to the DFDs including the handling of aliases, definition of data structures and the implementation of the data dictionary.
- Next, the processes or operations in the DFDs have to be defined by analyzing the process logic and then depicting them in pseudocode.
- Finally, a logical, hierarchical design needs to be derived from the flat DFDs. This stage begins with evaluating the control mechanism, the changeability of the modules, modular cohesion and coupling, transform analysis, design refinement, error and exception handling, and transaction analysis.

Design begins with an evaluation of the higher level DFDs, the type of information flow is identified and the flow boundaries that separate the transform or transaction centers are defined. The transforms are then mapped to the program structure as modules.

Tools in Structured Design: Structure Chart

- Shows the partitioning of software into modules, how the modules are organized and what information modules pass to one another
- Concentrates on “black box” aspect of modules, their external functions, inputs and outputs rather than their internal procedures and data

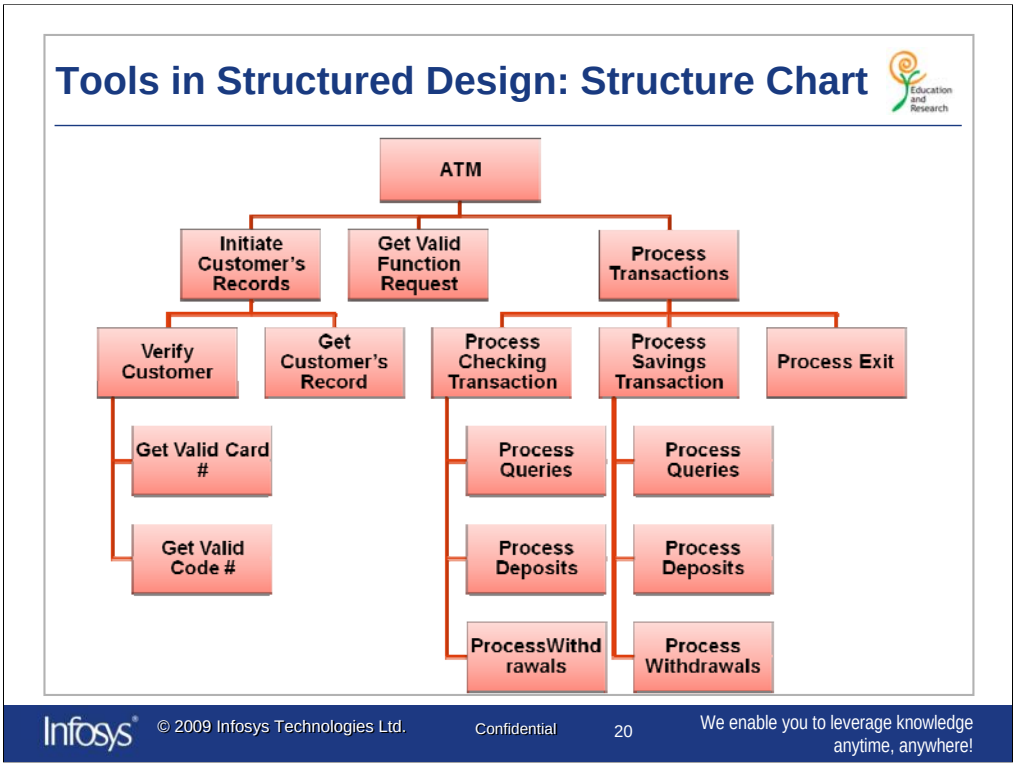
Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

A structure chart is a top down modular design tool, constructed of squares representing the different modules in the system, and lines that connect them. The lines represent the connection and or ownership between activities and sub activities as they are used in organization charts.

A structure chart depicts

- the size and complexity of the system, and
- number of readily identifiable functions and modules within each function and
- whether each identifiable function is a manageable entity or should be broken down into smaller components.

A structure chart is also used to diagram associated elements that comprise a run stream or thread. It is often developed as a hierarchical diagram but other representations are allowable. The representation must describe the breakdown of the configuration system into subsystems and the lowest manageable level



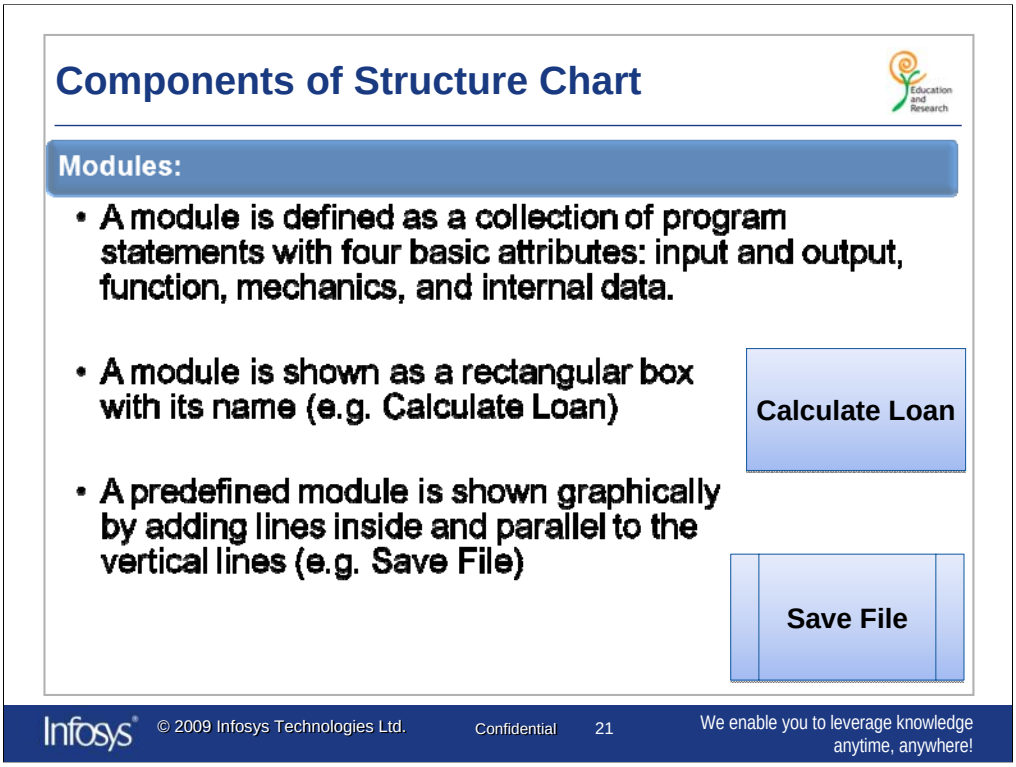
A structure chart can be developed starting with the

• Creation of a structure, which places the root of an upside-down tree which forms the structure chart.

• The next step is to conceptualize the main sub-tasks that must be performed by the program to solve the problem.

•Next, the programmer focuses on each sub-task individually, and conceptualizes how each can be broken down into even smaller tasks.

•Eventually, the program is broken down to a point where the leaves of the tree represent simple methods that can be coded with just a few program statements



Each module must have a module name.

Module names should consist of a transitive (or action) verb and an object noun. Module names and numerical identifiers may be taken directly from corresponding process names on Data Flow Diagrams or other process charts.

The name of the module and the numerical identifier is written inside the module rectangle. Other than the module name and number, no other information is provided about the internals of the module.

Components of Structure Chart contd..

Connections between modules

- Modules are organized into a hierarchy, cooperating and communicating to get the job done

Communication between modules

- Data that is exchanged between modules (see direction of arrows)

Infosys © 2009 Infosys Technologies Ltd. Confidential 22 We enable you to leverage knowledge anytime, anywhere!

Few ways of connecting the different modules:

Off-page connector :

A named object that essentially breaks a single call into two calls that are on different Structured Charts. *Off-Page Connectors* are associated with each other if they have the same name. The user can traverse from an *Off-Page Connector* on one Structured Chart to the associated *Off-Page Connector* on another Structured Chart (STC) at the same or a different level, directly from the diagram editor (DGE).

On-page connector :

A named object that essentially breaks a single call into two calls that are on the same STC. *On-Page Connectors* are associated with each other if they have the same name. The user can traverse from an *On-Page Connector* on one part of a STC to the associated *On-Page Connector* on another part of the STC directly from the DGE.

Communication between modules:

One module could interact with another module zero or more times at runtime.

One module can call another module and thereafter executes in an independent manner; used for processes and tasks.

One module can iteratively call another module, for example a call within a loop.

Deriving a good structure chart

Three steps involved in getting to the structure charts:

1. Break the system into suitably tractable units by means of "Transaction analysis"
2. Convert each unit into a good structure chart by means of "Transform analysis"
3. Link back the separate units into overall system implementation

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

Tools in Structured Design: Pseudocode

- Structure chart must be accompanied by description of each module appearing on it
- Pseudocode is one method of module specification
- It is an informal language similar to structured English
- It is used to organize a programmer's thoughts prior to coding
- It is tool of the designer and programmer rather than the user and analyst

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

Pseudocode is a compact and informal high-level description of a computer programming algorithm_that uses the structural conventions of a programming language, but is intended for human reading rather than machine reading.

Pseudo-code typically omits details that are not essential for human understanding of the algorithm, such as variable declarations , system-specific code and subroutines. The programming language is augmented with natural language descriptions of the details, where convenient, or with compact mathematical notation.

The purpose of using pseudocode is that it is easier for humans to understand than conventional programming language code, and that it is a compact and environment-independent description of the key principles of an algorithm. It is commonly used in textbooks and scientific publications that are documenting various algorithms, and also in planning of computer program development, for sketching out the structure of the program before the actual coding takes place.

Tools in Structured Design: Pseudocode

Example:

```
Module prod customers for pending bills
    /* scans the pending bills file and issues notices to customers for bill payment */
      Open pending bill file, customer details file
      Get current date
      Repeat
          call get next pending bill
            getting pending bill, end of pending bill file
      Until end of pending bill file
          set days overdue to current date – bill date
          case days overdue
            > 100: call generate legal threat using pending bill
            31-100: call generate stern warning using pending bill
            0 – 30: no action
          end case
      End repeat
      Close pending bill file, customer details file
End module
```


© 2009 Infosys Technologies Ltd.

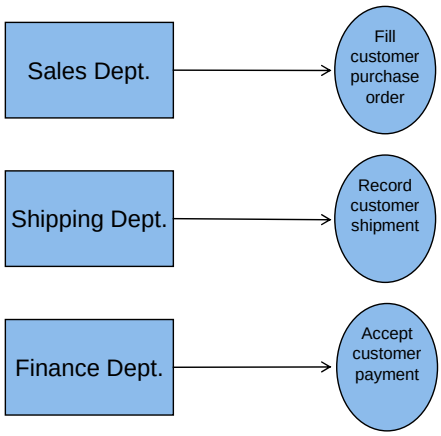
Deriving a good structure chart

- Three steps involved in getting to the structure charts:
  - Break the system into suitably tractable units by means of “Transaction analysis”
  - Convert each unit into a good structure chart by means of “Transform analysis”
  - Link back the separate units into overall system implementation

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

Transaction analysis Strategy

- Transaction analysis is the technique of identifying the transaction types of a system and using them as the units of design
- Example – An order processing system would have transactions such as *Filling customer purchase order, recording customer shipment, accepting customer payment*
- Each of these transaction would be designed separately by transform analysis



Infosys © 2009 Infosys Technologies Ltd. Confidential 27 We enable you to leverage knowledge anytime, anywhere!

Transaction analysis uses as input the system flowchart and the event table to develop the top level i.e the main program.

Transaction analysis is a design strategy useful for transaction processing programs. Generally a structure chart derived using this strategy will have a transaction center module at the higher level and below it will be multiple transaction modules.

Each module is responsible for each transaction type.

Two sets of measurements are used to evaluate the quality of design:

•Coupling

•Cohesion

Transform Analysis Strategy

- Transform analysis or transform-centered design is the strategy for converting into a structure chart each piece of the DFD that we isolate by transaction analysis.

Transform analysis is composed of the following five steps:

Drawing a DFD of a transaction type (usually done during analysis phase)	Finding the central functions of the DFD	Converting the DFD into a rough-cut structure chart	Refining the structure chart by means of structured design criteria	Verifying that the final structure chart meets the requirements of the original DFD

Infosys © 2009 Infosys Technologies Ltd. Confidential 28 We enable you to leverage knowledge anytime, anywhere!

Generate the first draft structure, based on the redrawn data flow, with the calling hierarchy and the required data couples. The organization of the structure chart very closely mirrors the data flow diagram from which it derives.

Add other modules as necessary to get input data via the user-interface screens, read from and write to the data stores and write output data or reports. These modules are lower-level or utility modules. Appropriate data couples are added based on the data flows to and from these data stores

From this point there is a less dependency on DFD.

Using any structured english or decision table documentation as a basis, add other required intermodule relationships such as looping and decision symbols.

Final refinements to the structure chart are done based on the quality control concepts

Transform analysis: Draw a DFD

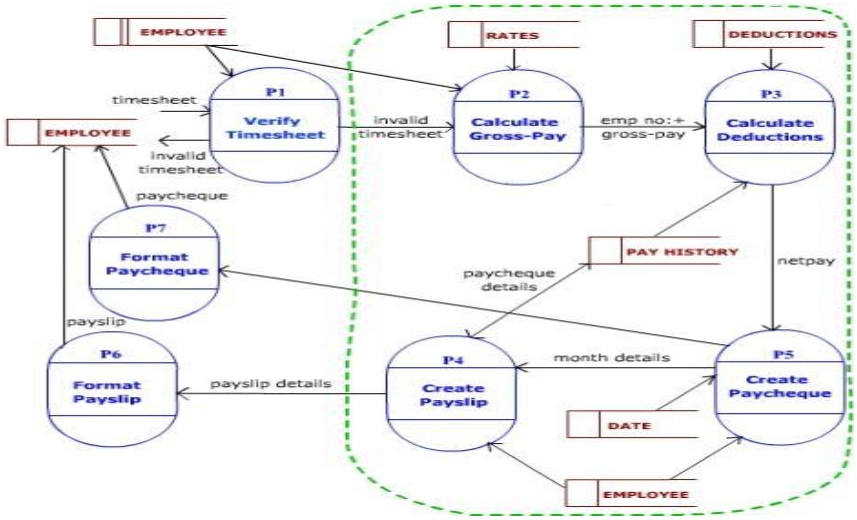
- If structured analysis has preceded Structured Design, then you already have your set of DFDs
- If not, then it is time to sketch out some DFDs as shown

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

DFDs can be used to provide both high level and detailed views of a system. The process of defining a system in a top down manner is called leveling a DFD.

A process can be uniquely identified by its corresponding number. The number is also an indicator of the location of that process.

Transform analysis: Draw a DFD



Transform analysis: Central Transform

Identifying the central transform

- Central transform is the portion of DFD that contains the essential functions of the system and is independent of the particular implementation of the input and output.
- To find the central transform, identify the center of the DFD by pruning off its afferent and efferent branches
  - Trace each afferent stream from the outside of the DFD towards the middle
  - Mark the data flow that represents the input in its most essential form
  - Trace each efferent stream from the outside of the DFD toward the middle
  - Mark the data flow that represents the output at its most essential form

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

The first step in transform analysis is to factor a DFD into afferent, central transform and efferent sections by considering the logical level of data flow within the DFD.

A high level structure chart is then created with one module to represent each afferent branch, each efferent branch, and central transform.

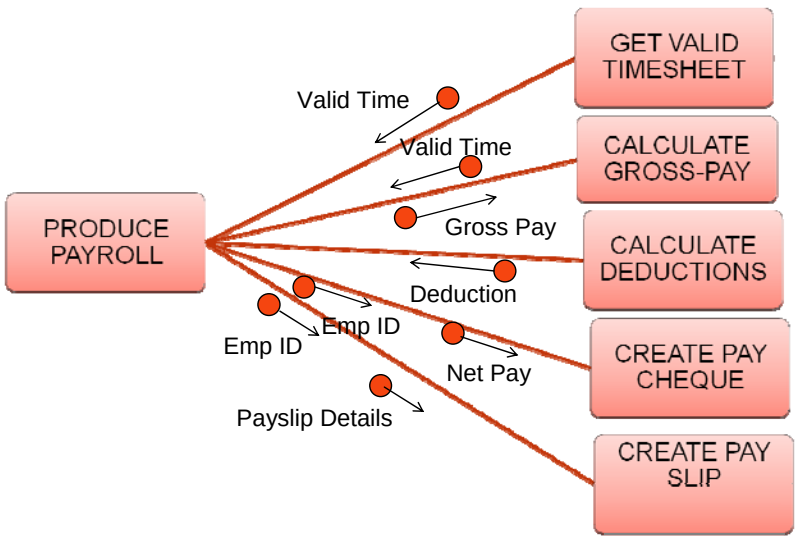
Within this initial chart , if necessary modules are then partitioned into sub components. Additional modules are often added to handle different operations.

Transform Analysis: First-cut structure chart

- The purpose of transform analysis is to turn a DFD for one transaction type into a structure chart for that transaction type.
- Structure chart depicts bosshood, no bosses in a DFD
- Pick up a boss (module) if there is a good candidate in the central transform, otherwise, hire a new boss
- DFD arrowhead won't necessarily correspond to structure chart arrowheads
- Direction of data flow isn't the same things as direction of call
- A module's name has to sump up the activities of its subordinates, while a bubble's name describes only its own activity
- Add read and write modules

© 2009 Infosys Technologies Ltd.

Transform Analysis: First-cut structure chart



Transform analysis: Revise the structure chart

- Add read and write modules for accessing sources, sinks, and files
- Factor and reorganize the afferent and efferent modules
- Factor the central transform, if necessary, using the DFD as a guideline
- Add error-handling modules
- Add initialization and termination details, if required
- Ensure that all modules have names in keeping with their hierarchical roles
- Show all flags that are necessary on a structure chart; for example, “end of stream” information
- Check all the design criteria and improve the design in keeping with those criteria

Infosys © 2009 Infosys Technologies Ltd. Confidential

We enable you to leverage knowledge anytime, anywhere!

Add initialization and termination details, if required

Ensure that all modules have names in keeping with their hierarchical roles

Show all flags that are necessary on a structure chart; for example, “end of stream” information

Check all the design criteria and improve the design in keeping with those criteria

Transform analysis: Verify Structure chart

The structure chart should implement the specifications correctly and live up the criteria for a maintainable system

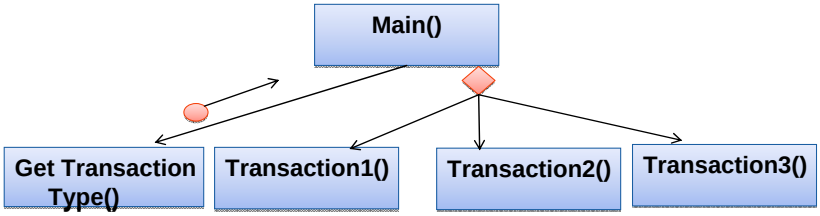
Get the structure chart reviewed by the author of the DFD to ensure that the structure chart correctly implements the requirements of the DFD

Infosys © 2009 Infosys Technologies Ltd. Confidential

We enable you to leverage knowledge anytime, anywhere!

Reconstructing the system

- Put together the collection of structure charts to form a complete system
- Tie together all of the structure charts for the transaction types under a transaction monitor or a transaction-center module
- Each structure chart can be “fired up” each time a transaction stimulus of its type arrives



Guidelines for a good decomposition of structure chart

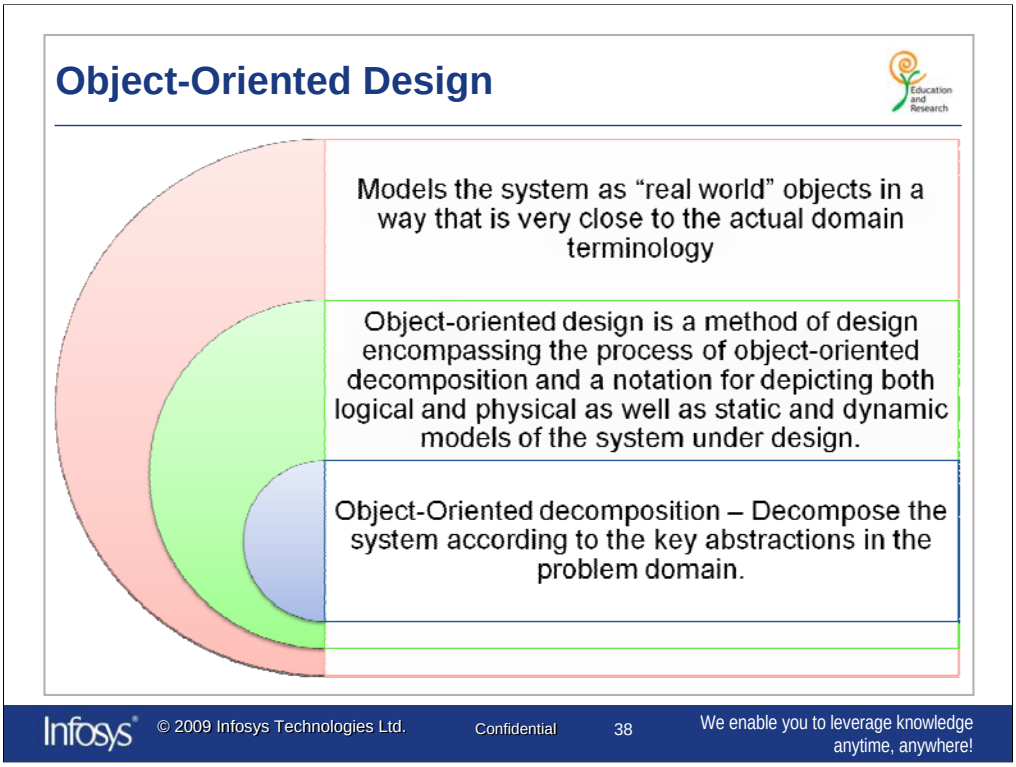
A module can affect only those modules which comes under it's control (All sub-ordinates, immediate ones and modules reporting to them, etc.)

Error should be reported from the module that both detects an error and knows what the error is.

Restrict fan-out (number of subordinates to a module) of a module to seven. Increase fan-in (number of immediate bosses for a module). High fan-ins (in a functional way) improves reusability.

Infosys © 2009 Infosys Technologies Ltd. Confidential

We enable you to leverage knowledge anytime, anywhere!

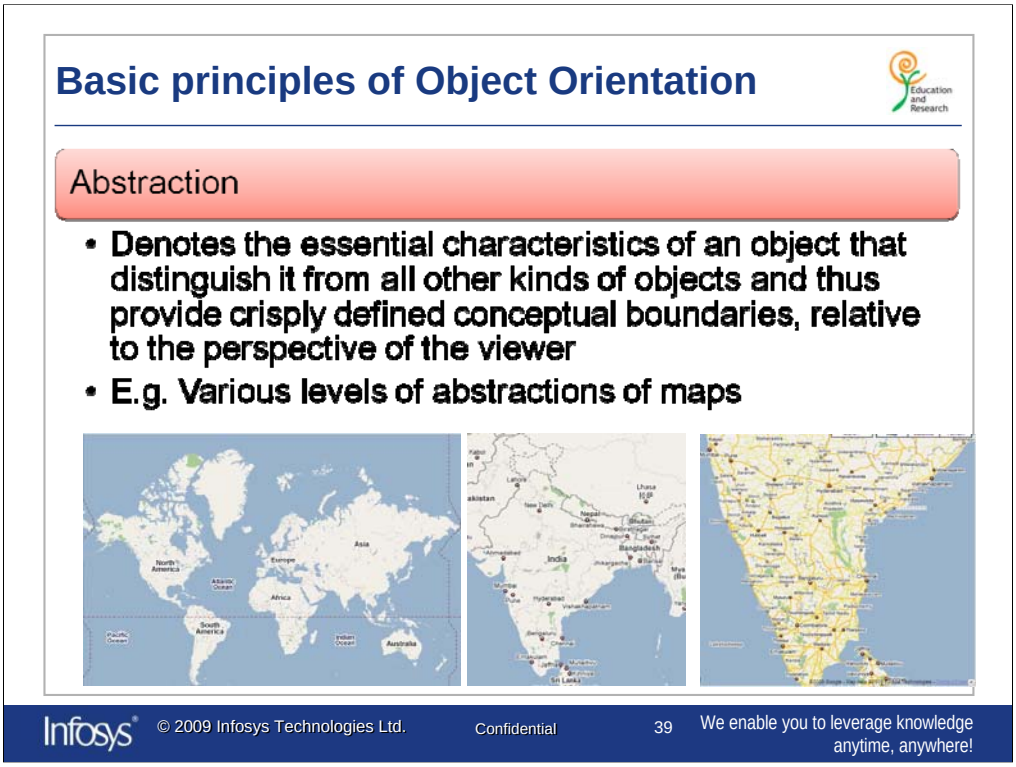


Object-oriented design is the process of planning a system of interacting objects for the purpose of solving a software problem. It is an approach to software design.

An object contains encapsulated data and procedures grouped together to represent an entity. The 'object interface', how the object can be interacted with, is also defined. An object-oriented program is described by the interaction of these objects. Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.

From a business perspective, Object Oriented Design refers to the objects that make up that business.

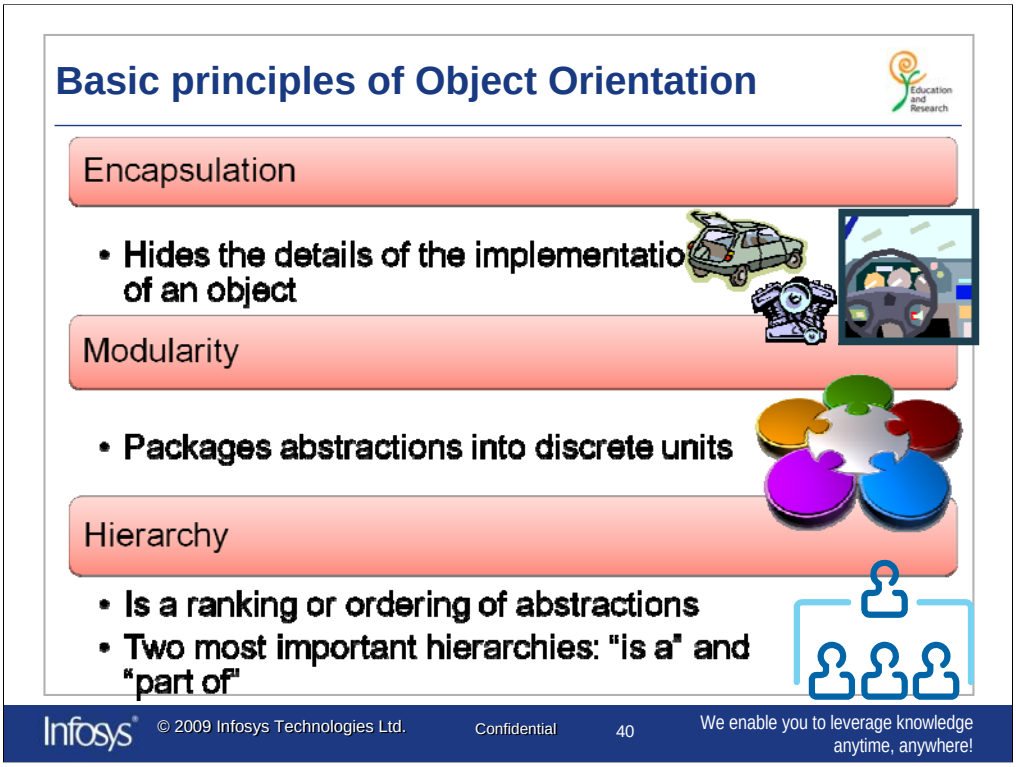
The input for object-oriented design is provided by the output of object-oriented analysis. An output artifact does not need to be completely developed to serve as input of object-oriented design; analysis and design may occur in parallel, and in practice the results of one activity can feed the other in a short feedback cycle through an iterative process. Both analysis and design can be performed incrementally, and the artifacts can be continuously grown instead of completely developed in one shot.



•Abstraction:

The process of exposing the relevant details and hiding the irrelevant details is called Abstraction. Abstraction is not an entirely new concept introduced by OOP. Structured Programming also supported abstraction with the help of data structures and functions. For example, one need not know the internal details of the printf function of C to use it. One need not know about how characters are converted to ASCII code and stored in the memory for using a character array.

OOP provides better abstraction. While structured programming provides abstraction at the level of data structures and function, OOP provides abstraction at a higher level; that is at the level of objects. Higher the abstraction level easier it is to understand.



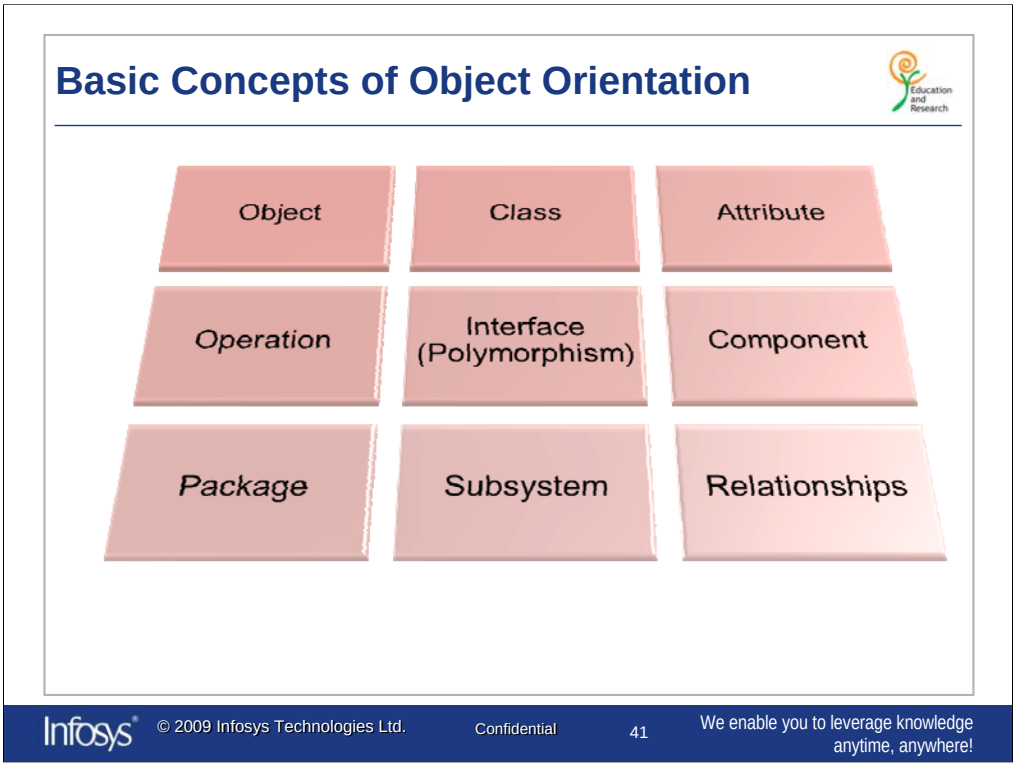
•Encapsulation

Encapsulation is the process of binding the data and the related functions together in a module and preventing other modules from directly accessing the data . This is clearly better than global data in structured programming which can be access by any module

•Inheritance(Hierarchy)

It is the mechanism which allows a class A to inherit properties of a class B. We say “A inherits from B". Objects of class A thus have access to attributes and methods of class B without the need to redefine them.

- Inheritance also promotes **reuse**. You don't have to start from scratch when you write a new program. You can simply reuse an existing repertoire of classes that have behaviors similar to what you need in the new program.



There are several concepts that object-oriented analysis is based on.

Some of these concepts require a totally new way of thinking about systems and the development process.

Object:

One important point to note about an object is that it has a state and a behavior.

Object can be Physical entity such car, animal, or it can be a Conceptual entity like a business process or it can even be a software entity like a Linked List

Attribute:

State of an object is nothing but a well defined condition of an item and behavior is observable effects of operation or event. For e.g. if you take an object, car – state will be current speed, current gear, engine state etc.

Operation:

Behavior will be slow down, accelerate, stop etc. State is internal to the object and it will not be exposed to the outside world.

Interface:(Runtime polymorphism)

Defining the prototype of all the operations of the class , leaving the definition/implementation to the user of the class. Interfaces are very useful when an unrelated set of classes have a common set of methods. Interfaces supports runtime polymorphism.

Package:

Packages are way of grouping a number of related classes and interfaces together into a single unit

•Organize your classes into smaller units and make it easy to locate and use the appropriate class file – i.e.,

you can split up the classes logically.

•Avoid naming conflicts – Hierarchical approach to store files/classes.

Please refer to notes page

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

•Packages allow you to protect your classes, data and methods in a larger way than on a class-to-class basis.

•Package names can be used to identify your classes.

•Relationships:
There are three kinds of relationships between classes :
Generalization and specialization (is-a relationship)

A class is similar to another class

Class is a different type of another

Aggregation (has-a relationship)

Class contains another class (as member)

Another class is part of the class

Association (uses-a relationship)

Loosely coupled relationship

A class interacts with another class

Object-oriented approaches to systems development are concerned with identifying attributes that are of interest regarding an object. It is important to note that with advances in technology, attributes have evolved to include more than simple data characteristics. Today, objects may include newer attribute types, such as a bitmap or a picture sound, or even video.

Let us understand the terms objects, instances of objects, and their attributes by an example. Take a payroll system, we need to maintain details of the salaries that are paid to employees. In this case, there will be salary detail for every person. This means that in this case, we all have same properties like name, basic pay, etc. for which data might differ from case to case. The behavior for this will also be the same, as salary slip might have to be generated, salary needs to the calculated etc. So, irrespective of the employee is me or the other person, we have same properties, common methods and relationships. So, instead of referring to individual objects, we can all of them in to one single class called Employees. Names and other details of each person, which needs to be captured will be an instance and it will be stored against name, basic salary etc. which will be the attributes

Please refer to notes page

Infosys © 2009 Infosys Technologies Ltd. Confidential 43 We enable you to leverage knowledge anytime, anywhere!

We can say Class is a blueprint used to create objects. In other words it is a software template that defines the methods and variables to be included in a particular kind of Object. Objects and classes do not exist in isolation. The things they represent interact with and impact one another to support the business mission.

Attribute is the data that represent characteristics about an object. This helps to determine the state of an object. For e.g. A Housing Loan class in a typical banking scenario may have attributes like Customer ID, Loan Period, Interest Rate, Loan Amount, Type of Interest (floating / fixed), etc.

Operation is nothing but a function that applies to attributes of a class. E.g. Calculate EMI may be a method inside a Loan class.

Behavior is the observable effects of an operation or event, including its results. In object-oriented circles, an object's behavior is commonly referred to as a method, operation, or service. For e.g. Behaviors for the Housing Loan class may be, Get Customer Details, Get (or Set) Loan Period, Get (or Set) Interest Rate, Get (or Set) Loan Amount and so on.

An Interface is a set of behaviors (a set of operations) offered by a class to the external world i.e.. other objects. E.g. Car provides interface like Break, Accelerate, Shift Gear, etc. to the driver. Though it has many internal functions, all those are not exposed to outside world.

Messages are implemented as methods to help collaborate with other objects. Sending and receiving messages from and to other objects is nothing but calling the appropriate methods of objects. The object sending a message does not need to know how the receiving object is organized internally or how the behavior is to be accomplished, only that it responds to the request in a well-defined way. For e.g. A Reporting class may pass a message to a Housing Loan object to Get Loan Period. (syntax can be: TheHousingLoan.GetLoanPeriod)

The OO Design Process

- In this phase, emphasis shifts from application domain concept towards implementation
- OOD consists of
  - Designing classes
    - Entity Classes
    - Classes other than Entity:
      - Boundary Classes
        - » UI Classes
        - » DB Classes
        - » Interface Classes
      - Control Classes
  - Applying design axioms
  - Iterating and refining the Physical Object Model
  - Designing the Access Layer classes (i.e. Physical Data Model)
  - Designing the View Layer classes (and UI prototype)
- Ready for Coding !

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

OOD process consists of:

• Designing classes which involves identifying and categorization of Entity Classes, Classes other than Entity like Boundary Classes, UI Classes, DB Classes, Interface Classes, Control Classes

• Applying design axioms – It aims at minimizing the information content of the design and also maintain the independence of components

• Iterating and refining the Physical Object Model

• Designing the Access Layer classes (i.e. Physical Data Model)

• Designing the View Layer classes (and UI prototype)

Then it is ready for coding.

SOFTWARE DESIGN ISSUES

© 2009 Infosys Technologies Ltd.

Software Design Issues

A number of key issues must be dealt with when designing software

Some are quality concerns that all software must address – for example, performance

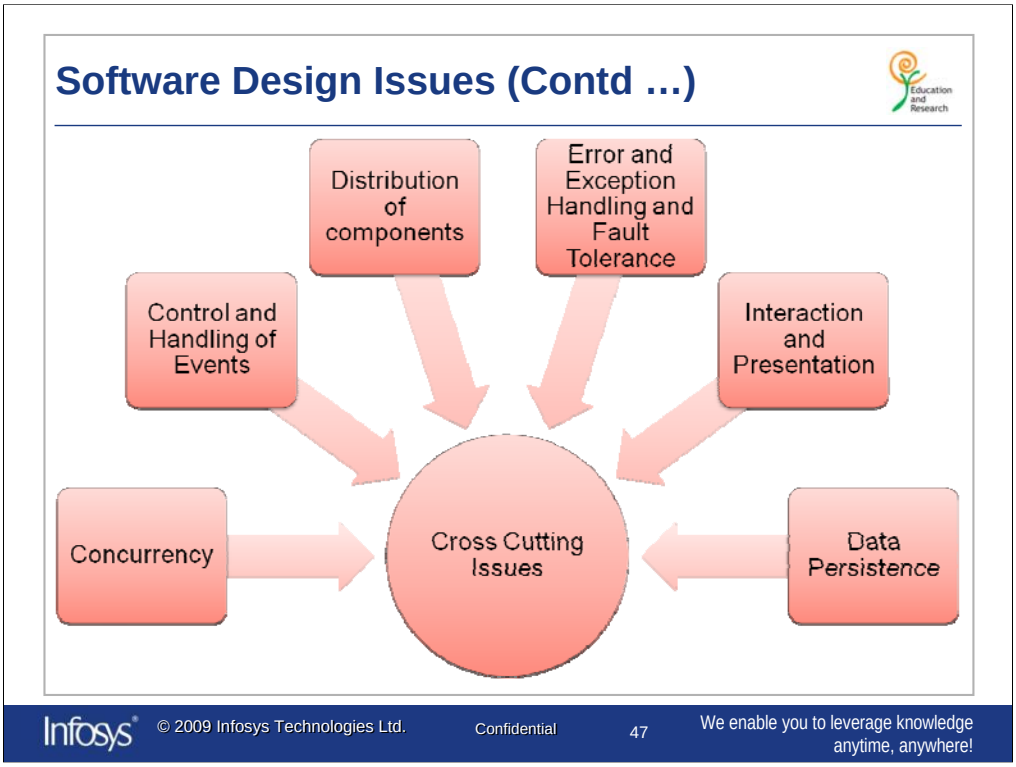
Another important and fundamental issue is how to decompose, organize, and package software components

In contrast, other issues "deal with some aspect of software's behavior that is not in the application domain, but which addresses some of the supporting domains."

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

Issues, which often cross-cut the system's functionality, have been referred to as aspects:

- Aspects tend not to be units of software's functional decomposition, but rather to be properties that affect the performance or semantics of the components in systemic ways.



Concurrency

How to decompose the software into processes, tasks, and threads and deal with related efficiency, atomicity, synchronization, and scheduling issues.

Control and Handling of Events

How to organize data and control flow, how to handle reactive and temporal events through various mechanisms such as implicit invocation and call-backs.

Distribution of Components

How to distribute the software across the hardware, how the components communicate, how middleware can be used to deal with heterogeneous software.

Error and Exception Handling and Fault Tolerance

How to prevent and tolerate faults and deal with exceptional conditions.

Interaction and Presentation

How to structure and organize the interactions with users and the presentation of information (for example, separation of presentation and business logic using the Model-View-Controller approach). It is to be noted that this topic is not about specifying user interface details, which is the task of user interface design (a part of Software Ergonomics)

Data Persistence

How long-lived data are to be handled.

Architecture Vs. Design

Architecture	Design
Architecture sheds non-functional decisions and partition functional requirements	Design is a principle through which functional requirements are achieved
Architecture is associated with the selection of architectural elements, the interactions among them and the constraints on those elements and their interactions	Design is to do with the modularization and detailed interfaces of the design elements, their algorithms and procedures, and the data types needed to support the architecture and to satisfy the requirements

© 2009 Infosys Technologies Ltd.

SOFTWARE DESIGN NOTATIONS, LANGUAGE, TOOL

© 2009 Infosys Technologies Ltd.

Software Design Notations

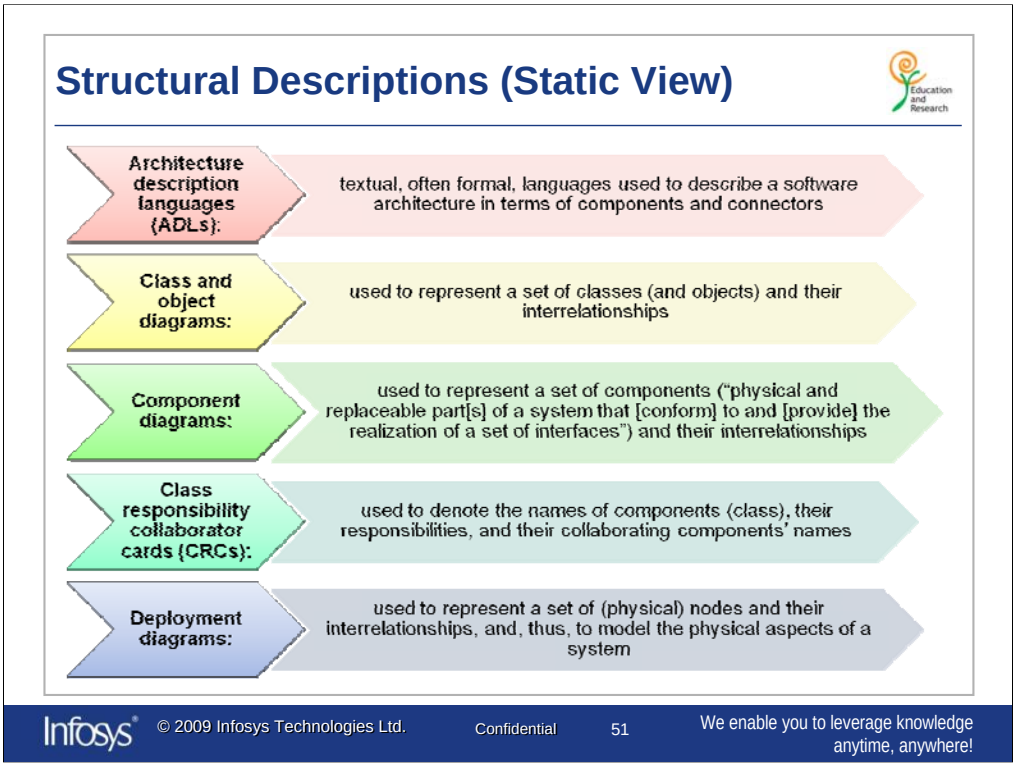
Software Design notations are used

- To describe a design's structural organization and to represent software behavior.
- During architectural design and during detailed design.
- Mostly in the context of specific methods (Structure Design, OO Design, etc.)

Design notations are categorized into notations for describing

- Structural (static) view
- Behavioral (dynamic) view

© 2009 Infosys Technologies Ltd.



Class diagram is the set of classes, interfaces, and collaborations and their relationships

Captures the vocabulary of the system

Build and refined throughout development

Purpose

- Model static design view of the system
  - Model vocabulary of the system
  - Specify collaborations
  - Specify logical database schema

Developed by analysts, designers, and implementers

Object diagram is a set of objects and their relationship at a point in time

Captures objects and links

Describes the static (logical) structure of a system at a particular time

Purpose

- Model data/object structures
- Specify static snapshots

Component diagram shows the organizations and dependencies among a set of components built as part of architectural specification

Represents the static implementation view of a system

Purpose

- Organize source code
- Construct an executable release
- Specify a physical database

Deployment diagram is the configuration of run-time processing nodes (and the components there in) and their relationships

Captures the topology of the hardware where the system executes

Purpose:

- Specify the distribution of components

Structural Descriptions (Static View)

Entity-relationship diagrams (ERDs):	used to represent conceptual models of data stored in information systems
Interface description languages (IDLs):	programming like languages used to define the interfaces (names and types of exported operations) of software components
Jackson structure diagrams:	used to describe the data structures in terms of sequence, selection, and iteration
Structure charts:	used to describe the calling structure of programs

Infosys © 2009 Infosys Technologies Ltd. Confidential

We enable you to leverage knowledge anytime, anywhere!

Behavioral Descriptions (dynamic view)

Activity diagrams	• Used to show the control flow from activity
Collaboration diagrams	• Used to show the interactions that occur among a group of objects, where the emphasis is on the objects, their links, and the messages they exchange on these links
Data flow diagrams (DFDs):	• Used to show data flow among a set of processes
Decision tables and diagrams:	• Used to represent complex combinations of conditions and actions
Flowcharts and structured flowcharts	• Used to represent the flow of control and the associated actions to be performed

Infosys © 2009 Infosys Technologies Ltd. Confidential

We enable you to leverage knowledge anytime, anywhere!

Activity Diagrams:

An activity represents an operation on some class in the system that results in a change in the state of the system

Typically used to model workflow or business processes and internal operations

“Swim-lanes” group related activities into one column

May be created for use-cases, classes, and/or operations

Purpose

Model workflow

Model operations

Collabration Diagrams:

Represent interactions between objects as a series of sequenced messages

Emphasizes the structural organization of the objects sending and receiving the messages

Captures dynamic behavior (message-oriented)

Easier to depict complex procedural logic or multiple parallel transactions

Purpose

Model flow of control over the structural organization of the objects

Behavioral Descriptions (dynamic view)

Sequence diagrams	• Used to show the interactions among a group of objects, with emphasis on the time ordering of messages
State transition and state chart diagrams	• Used to show the control flow from state to state in a state machine
Formal specification languages:	• Textual languages that use basic notions from mathematics (for example, logic, set, sequence) to rigorously and abstractly define software component interfaces and behavior, often in terms of pre- and post-conditions
Pseudocode and program design languages (PDLs)	• Structured-programming-like languages used to describe, generally at the detailed design stage, the behavior of a procedure or method

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

A use-case diagram is a diagram that shows set of use cases and actors and their relationship

Captures system functionality as seen by users

Built in early stages of development

Purpose:

- Specify the context of a system
- Capture the requirements of a system
- Validate a system's architecture
- Drive implementation and generate test cases

State diagram:

Models the behavior of a single object over its lifetime, specifying the sequence of events an object goes through in response to events

Especially useful in modeling reactive objects whose states are triggered by specific events

Actions within each state – syntax: do/ entry/ exit/

Purpose

- Model object lifecycle
- Model reactive objects (user interfaces, devices, etc.)

Unified Modeling Language (UML)

What is UML?

A Modelling language for visualizing, specifying, constructing, and documenting a software system and its components

Infosys © 2009 Infosys Technologies Ltd. Confidential 55 We enable you to leverage knowledge anytime, anywhere!

The Unified Modeling Language (UML) is a graphical language for visualizing, specifying and constructing the artifacts of a software system. The UML is used to detail the artifacts in the system, to document and construct - it is the language that the blueprint is written in. The UML may be used in a variety of ways to support a software development methodology (such as the Rational Unified Process, RUP) - but in itself it does not specify that methodology or process.

It is programming and language independent. It provides several diagrams that help to articulate the application design. UML is more than just a bunch of graphical symbols and creating models in UML facilitates communication.

UML:

- Provides ready-to-use, expressive and visual modeling language to modelers, to create meaningful models
- Helps extend core concepts by furnishing extensibility and specialization mechanisms
- Supports specifications that are independent of any programming language and development process
- Offers a formal basis for understanding the modeling language (thereby encourages more modeling tools)
- Supports higher-level developmental concepts like framework, collaboration, components and patterns.

UML - Features

- Visualizing
  - UML is a graphical language
  - Notations of UML are visual and well-defined
- Specifying
  - UML helps build models that are precise, unambiguous, and complete

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

UML is a graphical language and the notations of UML are visual and well-defined. This ensures consistent application of models and modeling elements. This facilitates easy adoption and use of modeling technique. While UML provides a comprehensive standard for modeling applications, it may not be exhaustive which no modeling method would provide. However, it furnishes extensibility and specialization mechanisms to extend the core concepts.

UML helps build models that are precise, unambiguous, and complete. Using different modeling notations to specify various relationships, links, forms and other details in a model ensures these characteristics. For e.g. though association, aggregation and composition are different types of relationships, all THREE have distinct notations to show them.

UML – Features (Contd …)

- Constructing
  - UML is not a visual programming language
  - However, its models can be directly connected to a variety of programming languages
  - It is possible to map from a model in the UML to:
    - OO Languages such as Java, C++, or C#
    - Tables in a relational database
    - The persistent store of an object-oriented database (e.g. Cache, Jasmine)

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

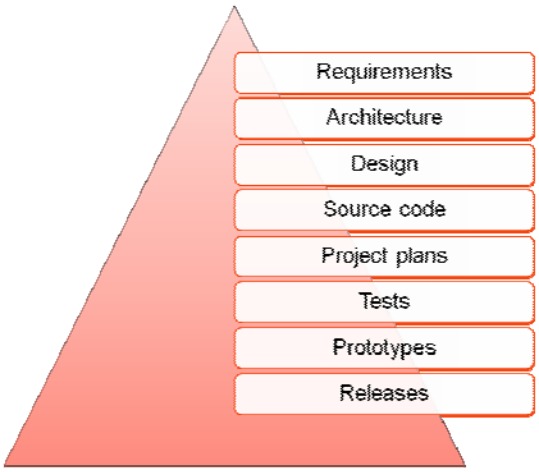
UML is not a visual programming language nor it provides any IDE (Integrated Development Environment) like Visual C++ to develop any programs using it. However, its models can be directly connected or mapped to a variety of programming languages. It is possible to map from a model in the UML to:

• OO Languages such as Java, C++, or C#
• Tables in a relational database
• The persistent store of an object-oriented database (e.g. Cache, Jasmine)

This mapping permits forward engineering – the generation of code from a UML model into a programming language. Reverse engineering is also possible with human intervention. Reverse engineering in this context means, that the UML tool reads program source code as input and derives model data and corresponding graphical UML diagrams from it. These two include round-trip engineering, meaning the ability to work in either a graphical or a textual view. Round-trip engineering refers to the ability of a UML tool to perform code generation from models, and model generation from code (a.k.a., reverse engineering), while keeping both the model and the code semantically consistent with each other.

UML – Features (Contd …)

The UML addresses the documentation of a system's architecture and all of its details



Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

The UML addresses the documentation of a system's architecture and all of its details throughout the software development life-cycle.

Listed here are the various SDLC stages and some of the UML elements contributing to the documentation need in that stage:

•Requirements: Use-case models, class diagrams

•Architecture: Component diagrams, Deployment diagrams

•Design: Class diagram, Sequence diagram, State machine

•Source code: Tools providing code generation feature from the models

•Project plans: Use-cases can be used for estimation purposes, Package diagrams for modularizing

•Tests: Use-cases can be used for preparing system test cases

•Releases: Use-case diagrams, Deployment diagrams

The Value of UML

- Is an open standard
- Supports the entire software development lifecycle
- Supports diverse applications areas
- Is based on experience and needs of user community
- Supported by many tools
  - Rational Rose
  - Rational XDE
  - Borland Together
  - Rational Software Architect
  - Power Designer

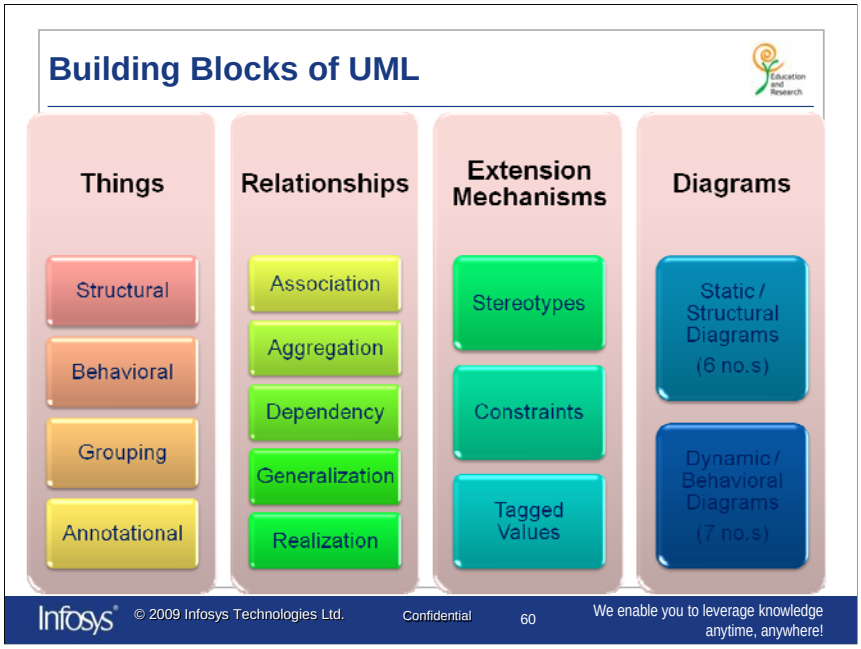
List of UML tools and their comparison is available in:
http://en.wikipedia.org/wiki/List_of_UML_tools

Infosys © 2009 Infosys Technologies Ltd. Confidential We enable you to leverage knowledge anytime, anywhere!

As UML is programming-language independent, it has become a *standard* modeling language. (UML modeling tools from IBM Rational are used extensively in J2EE shops as well in .NET shops.) Also, the UML notation set is a language and not a methodology, process or a procedure. This is important, because a language, as opposed to a methodology, can easily fit into any company's way of conducting business without requiring change. Consequently, it supports diverse application areas.

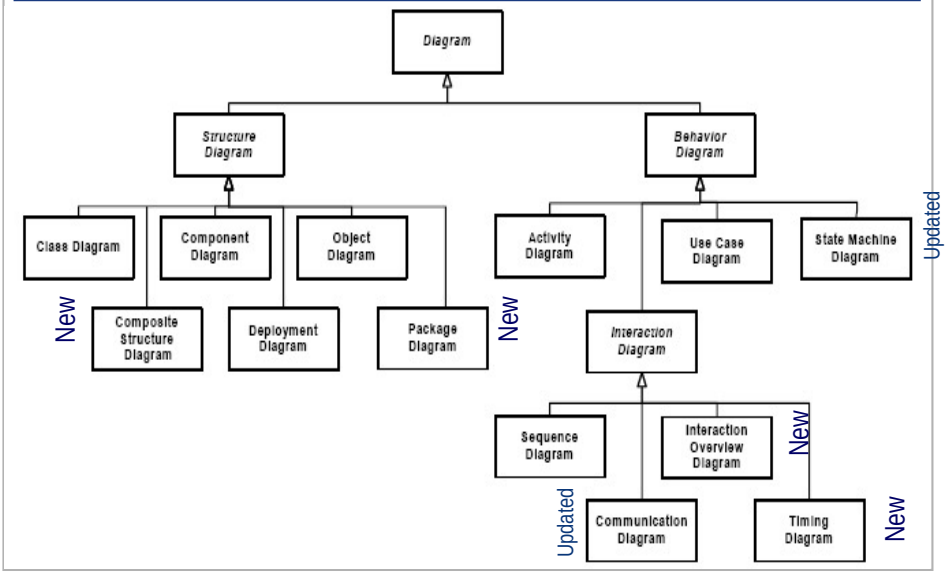
UML is not developed by 1 or 2 individuals nor by a company. It is based on several years of experience of many researchers, companies and various users. Best practices of various modeling techniques has been distilled into it.

Since UML is not a methodology, it does not require any formal work products (i.e.., "artifacts" in IBM Rational Unified Process® lingo). Yet it does provide several types of diagrams that, when used within a given methodology, increase the ease of understanding an application under development.



UML is a modeling language. Models are mostly expressed in terms of diagrams. A modeling diagram typically contains few blocks or boxes, lines connecting them and some text to add more information to the model. These are nothing but things, relationships and extensions in UML terms. Subsequent slides discuss these terms in detail.

UML 2.0 Diagram Taxonomy



This diagram shows the classification of UML diagrams.

Summary

- Fundamentals of Software Design
- Software Design Concepts
- Software Design Considerations
- Software Design Process
  - High-level Design
  - Detailed Design
- Software Design Strategies and Methods
- Software Design Issues
- Software Design Notations
- Software Design Tools

© 2009 Infosys Technologies Ltd.

References

- L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*
- G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*
- D. Budgen, *Software Design*
- F. Buschmann et al., *Pattern-Oriented Software Architecture: A System of Patterns*
- Grady Booch, *object oriented analysis and design with applications*
- Gamma, Helm, Johnson, Vlissides, *Design Patterns*
- Page-Jones, *The practical guide to Structured Systems Design*
- Kshop link (click on the *paper* link mentioned in this blog: http://mtc/MTCBlogs/Lists/Posts/Post.aspx?List=77ac78dd%2D12ba%2D4f02%2Db804%2D9f7c6aa17e08&ID=497

© 2009 Infosys Technologies Ltd.

We enable you to leverage knowledge anytime, anywhere!

We enable you to leverage knowledge anytime, anywhere!

Education and Research

Thank You

Education & Research

© 2009, Infosys Technologies Ltd.