System Design

G.Murugeswari
Asso. Prof. of CSE Dept.
MS University

System Design

- Design:
  - Evolving the analysis model to Design Models
  - Packages and Package Diagrams
  - Design Strategies
  - Developing the Actual Design
- Object Design
  - Object Design Activities
  - Method Specifications
- Data Management Layer Design
  - Object-Persistence Formats
  - Designing Data Access and manipulation classes.

Design

Evolving the analysis model to Design Models

- Design:
  - Create a blue print
  - Examine design strategies
  - User Interfaces, input and outputs
  - Physical Architecture decisions are made
    - Hardware, software, Centralized, Distributed
  - Global Issues and Security

Design

Evolving the analysis model to Design Models

- Functional and Non-functional requirements are addressed.

Problem domain-oriented analysis models into Optimal solution domain oriented

1. Factoring
2. Partitions and collaborations
3. Layers

Evolving the analysis model to Design Models

Factoring

- Separating out a module into standalone module in and of itself.
  - Find out the similarities and apply generalization, aggregation relationships.
  - Abstraction, Refinement

Partitions and Collaborations

- Partition is the object oriented equivalent of a subsystem.
- Communication Diagram –Identifies collaborations
  - Coupling and cohesion

Evolving the analysis model to Design Models

Layers

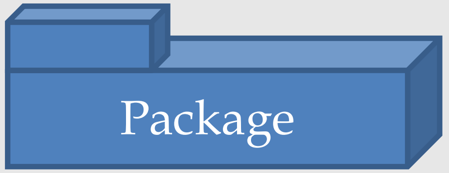
- Represents an element of the s/w architecture
- Other elements: User interface, data access, Management
    - Foundation : classes of data types , Abstract classes
        - Eg : Date, Time
    - Physical Architecture : Classes and communication between s/w and computer, middleware
    - Human Computer Interaction : User interface (Buttons, Windows)
    - Data Management storage and retrieval
    - Problem Domain : OO concepts

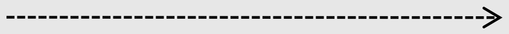
Package Diagram
Identifying Packages and Creating PD

Package Diagram

Logical Grouping of UML elements

Simplify UML Diagrams by grouping related elements into a single higher level elements

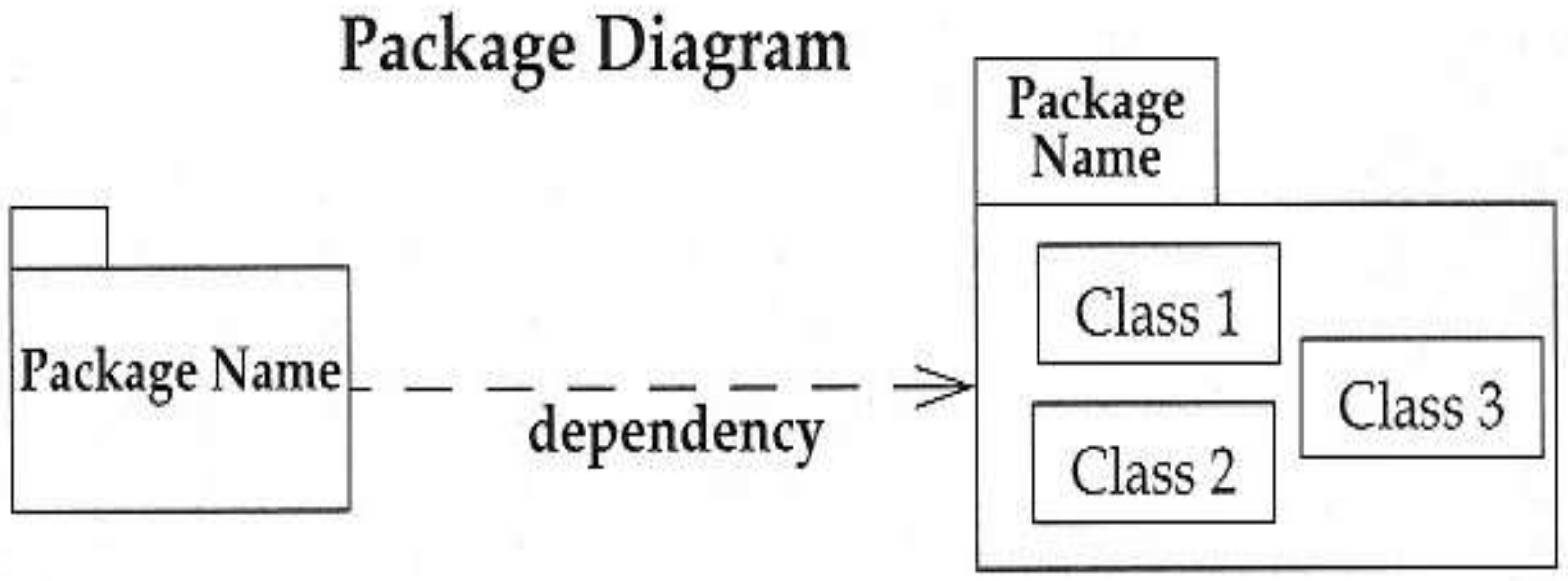




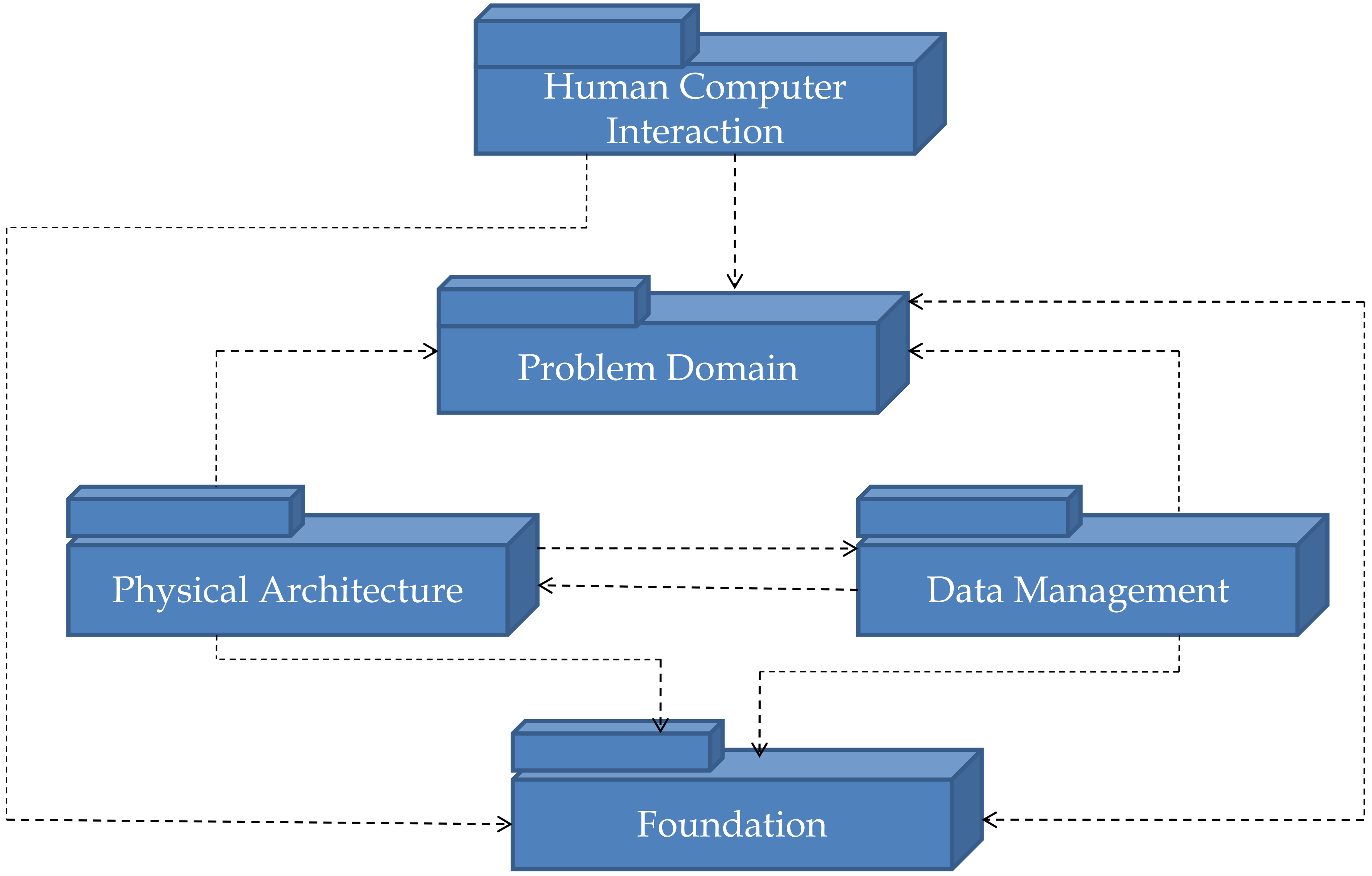
Dependency Relationship

Represents dependency between the packages
Arrow - > Drawn from the dependent package toward the package on which it is dependent

Package Diagram



Package Diagram



Package Diagram of Dependency Relationships among Layers

Package Diagram
Identifying Packages and Creating PD

I. Set the context
II. Cluster classes together based on shared relationships
III. Model clustered classes as package
IV. Identify the dependency relationships among packages
V. Place dependency relationships between packages

Design Strategies

- Custom development
- Packages software
- Outsourcing

Design Strategies
Custom development

- Custom development
  - Complete control over the system
  - Standard Technology, Interfaces
  - Technical skill and Functional Knowledge within the company
  - Understanding business grows
  - Dedicated effort (long hours , hard work)

Risk :
  - No Guarantee on success
  - Pulled away to work on other projects
  - Technical obstacles
  - Business users become impatient with growing timeline

Design Strategies
Packaged Software

- Packaged Software
  - “Reinvent the Wheel “
    - Eg. : Word Processor, Pay roll
  - Diff. Range of Reusable Components
    - ActiveX, Single Function tool – ERP Application
  - Packaged Applications allow for customization
    - Logo
  - “ **Workaround**” can be created if Amount of customization is not enough
    - Last resort - Not supported by vendors
  - **System Integration** - Combines Packaged s/w existing legacy systems and new software written.   Challenge : Data produced by different packages, Identifying Transformations
  - **Object Wrapper** – Object wraps around a legacy system

Design Strategies
Outsourcing

- Outsourcing : Hiring external vendor, developer or service provider.
  - Popular in recent years
  - Technology Experienced Developer, Resources Availability ,Low cost
  - Adds value to business
  - Lose control over future development

  Risk :

  1. Understanding Requirements
  2. Selection of Vendor

Design Strategies
Outsourcing

- Contract
  - Time and Arrangement
  - Fixed Price Contract
  - Value added contract

Design Strategies

	Use custom development when	Use a Packaged system when	Use Outsourcing when
Business Need	Unique	Common	Not core to the business
In-house Experience	In-house Functional and Technical Experience exists	In House Functional experience exists	In-house Functional and Technical Experience does not exists
Project Skills	Desire to build In-house skills	Skills are not strategic	Outsourcing is a strategic decision
Project Management	Has High skilled project manager and proven methodology	Project Manager coordinates vendor effort	Highly skilled manager matches the scope of outsourcing
Time Frame	Flexible	Short	Short or flexible

Developing Actual Design

- Alternative Matrix
  - Organizes pros and cons of the design methods
  - Combines flexibility analyses into one matrix

Alternative Matrix

	Alternative –I Shop-with-Me	Alternative –II Web Shop	Alternative –III Shop-N-Go
Technical Feasibility	C . C Experience in-house	C and Java Develop C and Java Skill	Java. Develop in-house Java Skills
Economic Feasibility	$150	$700 no yearly charges	$200/ Year
Organizational Feasibility	Pgm used by other companies	Pgm used by other companies	Few companies have experience with shop-n-go
Other Benefits	Simple to use	Tom had limited experience. Bit positive experience with this	
Other Limitations			Interface is not clearly customized

Object Design Activities

- Activities used to design classes and methods
  - Additional Specification
  - Identifying opportunities for reuse
  - Restructuring the design
  - Optimizing the Design
  - Mapping the problem domain classes to an implementation language

Object Design Activities
Additional Specification

- Ensure that the classes are necessary and sufficient
  - Ensure : No Missing Attribute/ Method, no extra , unused attribute or methods.
- Finalize the visibility
    - Eg. :Smalltalk : Attributes are hidden and methods are visible
- Decide on signature of every method in every class (Signature : Name of the method, parameter, return value)
- Define constraints that to be preserved by the object
    - Constraints : Pre, post, invariant Eg: : Age
    - Handling Violations of constraints

Object Design Activities

Identifying opportunities for Reuse

1. Design Patterns
2. Framework
3. Class Library
4. Component

Object Design Activities

Identifying opportunities for Reuse

- **Design Patterns**
  - Group of collaborating classes.
  - Interfaces
  - Eg. Command Processor pattern, Forwarder Receiver Pattern, Design patterns in C++ , Java

Object Design Activities

Identifying opportunities for Reuse

- Framework
  - Composed of set of implemented classes.
  - CORBA, DCOM Fm used in Physical arch.

Object Design Activities

Identifying opportunities for Reuse

- Class Library
    - set of implemented classes designed for reuse.
    - CL for Numerical, Statistical processing
    - Eg: Greatest Number uses Max function

Object Design Activities

Identifying opportunities for Reuse

- Component
    - Piece of s/w plugged into a system
    - Has well defined API
    - Implemented using Class lib. and framework.
    - Mainly used in problem domain

Object Design Activities

Restructuring the Design

- Factoring : Reviewing a set classes on layer.
- Normalization : Helps to identify missing classes
- To define relationship, associations and aggregations as attribute.
- Inheritance relationship for checking generalization/specialization

Object Design Activities

Optimizing the Design

Relationship between the efficiency and understandability

- ❖ Increasing the understandability will result in design that is more difficult to understand

- ❖ Review the access paths between objects
- ❖ Review each attribute of each class
  - ❖ Move the attribute to speedup the process

Object Design Activities
Optimizing the Design

- Review the direct and indirect fan-out
  - Optimize
- Look at execution of order of the statement
  - Searching an attribute
- Avoid re-computation
  - Total, Average

Method Specification

- Documents that include explicit instruction on how to write the code to implement the method
- Needs to be clear and easy to understand
- No formal syntax
- Components
  - General Information
  - Events
  - Message Passing
  - Algorithm Specification

Method Specification Form

Method Name:	Class Name:	ID:		
Contract ID:	Programmer:	Date Due:		
Programming Language:	Visual Basic	Smalltalk	c++	Java
Triggers/Events:				
Arguments Received: Data Type:	Notes:			
Messages Sent & Arguments Passed: ClassName.Method Name	Data Type:	Notes:		
Argument Returned: Data Type:	Notes:			
Algorithm Specification:				
Misc.Notes:				

Algorithm Specification in structured English

Common Statements	Examples
Action Statement	profits=Revenues – Expenses Generate Inventory-Report

If Statement	IF Customer Not in the Customer Object Store
	THEN Add Customer record to Customer Object Store
	ELSE Add Current-Sale to Customer's Total-Sales
	  UPDATE Customer record in Customer Object Store

For Statement	FOR all Customers in Customer Object Store DO Generate a new line in the Customer-Report Add Customer's Total-sales to Report-Total

Case Statement	CASE
	    IF Income<10,000: Marginal-tax-rate = 10 percent
	    IF Income<20,000: Marginal-tax-rate = 20 percent
	    IF Income<30,000: Marginal-tax-rate = 30 percent
	    IF Income<40,000: Marginal-tax-rate = 40 percent
	    ELSE Marginal-Tax-Rate = 38 percent
	ENDCASE

Pseudocode

```
(Get_CD_Info module)
    Accept (CD.Title) (Required)
    Accept (CD.Artist) (Required)
    Accept (CD.Category) (Required)
    Accept (CD.Length)
 Return
```

Data Management Layer Design

- Design of storage, Data Access and Manipulation logic
- How data is stored and handled by the programs that run the system
- For steps
  1. Selecting the format of storage
  2. Mapping the problem domain object to object-persistence format
  3. Optimizing the object-persistence format
  4. Designing the data access to handle commn. between the system and database.

Data Management Layer Design

- Design of storage, Data Access and Manipulation logic
- How data is stored and handled by the programs that run the system
- Four steps
  1. Selecting the format of storage
  2. Mapping the problem domain object to object-persistence format
  3. Optimizing the object-persistence format
  4. Designing the data access to handle commn. between the system and database.

Data Management Layer Design

- Design of storage, Data Access and Manipulation logic
- How data is stored and handled by the programs that run the system
- Four steps
  1. Selecting the format of storage
  2. Mapping the problem domain object to object-persistence format
  3. Optimizing the object-persistence format
  4. Designing the data access to handle commn. between the system and database.

Data Management Layer Design

Selecting the format of storage

Types :

Files (Sequential and Random Access)

Object –Oriented Databases

Object –Relational Databases

Relational - Databases

Data Management Layer Design

Files (Sequential and Random Access)

Ordered, Unordered

Types : Master file, Look-up file, Transaction file, Audit file and history files.

Data Management Layer Design

- Relational Databases
  - Keys, Constraints, Referential Integrity
  - SQL (Structured Query Language)
  - Eg.: MS Access, Oracle, DB2, SQL Server
  - Design: Class Diagram to Relational db schema

Data Management Layer Design

- Object-Relational DBMS
  - RDBMS with extension to handle the storage of Objects in relational table (User defined data types)
    - RDBMS : simple data type, ORDBMS : Complex (Map)
    - Eg.: DB2, Informix, Oracle (Extends the Support for Object)
  - Many ORDBMS does not support many of OO features (Eg: Inheritance). Different methods.

Data Management Layer Design

- Object-Oriented DBMS
  - 2 Approaches
    - Adding persistence extension to OO Language
    - Creation of entirely new DB
  - Defines ODL (Object Definition Language), OML (Object Manipulation Language), OQL (Object Query Language)
  - Objects associated with an extent (Object ID)
  - Supports for inheritance
  - Supported by C++, Java, Smalltalk
  - Supports Multi valued attributes, multimedia app.s , complex data (video, graphics and sound)
  - Eg.: Gemstone, Jasmine, O2, POET, Versant

Data Management Layer Design

	Sequential and Random Access	Relational DBMS	Object RDBMS	OO DBMS
Major Strengths	Part of OOPL Files can be designed for fast performance Good for Short term storage	Can handle diverse data needs Leader in DB market	Based on established technology SQL Handle complex data	Handle complex data Direct support for object orientation
Major Weakness	Redundant Data No query Language No access control	Cannot handle complex data No support for Object Orientation Impedance mismatch btwn table & object	Limited support for Object Orientation Impedance mismatch btwn table & object	Skills are hard to find Still Maturing Technology
Data Types Supported	Simple and complex	Simple	Simple and complex	Simple and complex
Types of App. Supported	Transaction Processing	Transaction Processing And decision Making	Transaction Processing And decision Making	Transaction Processing And decision Making
Existing Storage Formats	Organization Dependent	Organization Dependent	Organization Dependent	Organization Dependent
Future Needs	Poor future prospects	Good future prospects	Good future prospects	Good future prospects

- Reference Book :
  - Systems Analysis & Design with UML Version 2.0 An Object Oriented Approach– Alan Dennis,Barbara Haley Wixom,DavidTegarden, Wiley India