Design Techniques using UML 2.0

Instructor:
Sridhar D P

KnowledgeWorks IT Consulting Pvt. Ltd.,
No. 65, Sri Vinayaka Tower, 3rd Floor, 8th B Main,
Jayanagar 4th Block, Bangalore – 560011
+91 80 26630622, 22459941, 41533451
training@knowledgeworksindia.com
www.knowledgeworksindia.com

About KnowledgeWorks:

KnowledgeWorks philosophy is based on power of knowledge and value it brings to individuals and organization per se. With growing competition and tight economy, organizations are expected to deliver results within tight budget and schedule. Doing things right the first time is the need of the day. In view of these factors, availability of trained and experienced resources has become all the more imperative to have an edge over the competition.

KnowledgeWorks located in the Silicon Valley of India – Bangalore to address the pressing need for skilled resources in a short time. KnowledgeWorks specializes in providing high end technical training in the areas of emerging technologies which include Internet Technologies, Mobile Computing, Cloud Computing, to name a few. We also help clients in other technologies like Client / Server, Databases, Software Testing, ERP, Datawarehousing, Mainframes technologies, etc.

It is for our professional approach to Corporate Training that our clients repose confidence in us and come back to us for their training needs.

The company's strength lies in deep understanding of technologies and markets backed by wide experience in domestic and international sales and marketing.

OUR MISSION

"Quality in a service or product is not what you put into it. It is what the client or customer gets out of it."

- ✓ *We are driven by strong, ineradicable and deeply rooted values of customer service, steadily striving towards excellence.*
- ✓ *In today's highly competitive business environment, Organizational survival and growth are dependent upon attracting, recruiting, training and retaining quality people. We thrive on the belief that people are the most important asset of any Organization.*

For further details about our Upcoming Workshops please visit:
http://www.knowledgeworksindia.com/seminars.php

Title : **Design** Techniques using UML 2.0

Instructor / Consultant : Sridhar D.P , Co founder – Summit

Consulting . Innovation. Mentoring

www.summit.in 1

Summit - Instructor Profile

- 16+ yrs of Experience
- Co Founder at Summit & SSQuare Innovations
- Post Engineering in Computer Science worked with Manufacturing firm
- Started Summit – Strategy Consulting in 2004 – working with Fortune clients
- Continued Education at IISC on Innovation & IP
- Executive PG from IIM Bangalore

Areas of Interest

- Strategy Consulting
- Business Modeling
- UML
- Innovation
- Product Development
- Teaching
  - 500 + programs on Business Engineering , OOAD, UML.
- Contacts
  - sridhar@summit.in

www.summit.in

www.summit.in 2

Course Objective

- Design Techniques using UML for Specialists is a hands on Workshop designed for TCQ Specialists using the OMG's UML Framework, OOAD Methodology & Industry Best practices.
- These sessions are Case driven and ensures that Participants will be able to apply the concepts and master them using tools like Enterprise Architect / StarUML.
- An overview of the SysUML will also be provided for Participants to get a big picture of modelling in Systems Engineering domains.

www.summit.in 3

Intended Audience

- TCQ Specialists

www.summit.in 4

Course Content

Day 1	Introduction	Introduction - Cross words, Quiz and Expectation Setting	
Module 1	UML & Object Thinking	Object Thinking basics UML Origin UML 2.0 & SysUML	Staruml / Enterprise Architect
Module 2	UML in Requirements Definition Phase	Context diagrams Usecase Diagrams Swimlane Diagram Activity Diagram	
Module 3	UML in Analysis Definition Phase	Domain Analysis, CRC cards Analysis Class diagram Analysis Sequence Diagram Analysis Communication Diagram	
Module 4	UML in Architecture Definition Phase	Layer Diagram Package Diagram Subsystem Diagram	

www.summit.in 5

Course Content

Module 5	UML in Design Phase	Design Class diagrams Sequence diagrams & Subsystem diagrams Interaction Overview diagrams diagrams Design Design - Package Statechart diagrams Timing	
Module 6	UML in Implementation & Deployment Phase	Component diagram diagram Deployment	
1	Tools	StarUML or Enterprise Architect	
2	Case	Case 1 : Automated Home Lighting - Embedded Case Case 2: Payroll Application Development	
3	Prerequsites	Participants must have exposure to OOAD and OOP methodologies	

www.summit.in 6

Lets follow this ...

- Be on time
  - Don't skip sessions
  - Any emergencies , update your Instructor and Manager.
  - 2 Tea breaks + 1 Lunch break
- Class Etiquette
  - Keep your cell phone in silent mode. Care for others in your class room.
  - Make a list of unanswered questions and take it up with your instructor after class hours.

www.summit.in 7

Let's know each other

- Your name
- Team that you work for
- Interest / previous experience on the subject
- Anything else that you want to share !

www.summit.in 8

Module 1 : UML & Object Thinking

Consulting . Innovation. Mentoring

Standish Group Survey

- 29% of **software projects** in large enterprises succeeded (i.e., produced acceptable results that were delivered close to on-time and on-budget)
- 53% were “challenged” (significantly over budget and schedule)
- 18% failed to deliver any usable result.
- The **projects** that are in trouble have an average budget overrun of 56%.

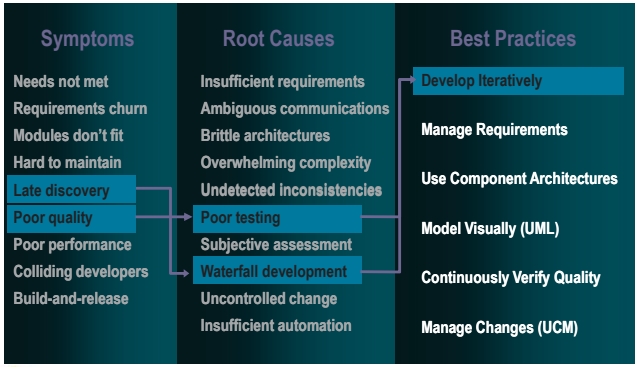
Standish Group

Year	On time, budget, original content	Delivered, but not according to original plan	Failed
2000	28%	49%	23%
1998	26%	46%	28%
1996	27%	33%	40%
1994	16%	53%	31%

IBM’s Insight into Project failure based on Standish Group Survey

- ✓ **User or business needs not met**
- ✓ **Requirements churn**
- ✓ **Modules don’t integrate**
- ✓ **Hard to maintain**
- ✓ **Late discovery of flaws**
- ✓ **Poor quality or end-user experience**
- ✓ **Poor performance under load**
- ✓ **No coordinated team effort**
- ✓ **Build-and-release issues**

Apply Best Practices



Best Practice to Tool Mapping

	Best Practice	Discipline	Tool	Other Tools
1	Develop Iteratively	Process	RUP /Open UP	XP / Scrum
	Adapt the Process			
2	Manage Requirements	Req Mgmt	Requisite Pro	
	Balance Stakeholder Priorities			
3	Component Architecture	Arch, Anal& Design	RSA / RSM	Staruml / Visio
	Elevate Level of Abstraction		Rose / XDE	
4	Model Visually	Notation	UML	UML
	Collaborate across Teams			
5	Continuously Verify Quality	Testing	Rational Test Suite	Mercury Suite
6	Manage Change	Change Mgmt	Clear Quest	Excel / freeware's
		Config Mgmt	Clear Case	Vss /Cvs
		Project Mgmt	Rat Portfolio Manager	MPP/

Concepts & Best Practices + Notation + Tool

OOAD + UML + StarUML

IBM's Six best practices

80-20 Rule

- 80% of the engineering is consumed by 20% of the requirements. Strive to understand the driving requirements completely before committing resources to full-scale development. Do not strive prematurely for high fidelity and full traceability of the requirements.
- 80% of the software cost is consumed by 20% of the components. Elaborate the cost-critical components first so that planning and control of cost drivers are well understood early in the life cycle.
- 80% of the errors are caused by 20% of the components.
  - Elaborate the reliability-critical components first so that assessment activities have enough time to achieve the necessary level of maturity.
- 80% of software scrap and rework is caused by 20% of the changes.
  - Elaborate the change-critical components first so that broad-impact changes occur when the project is nimble.

80 – 20 Rule

- 80% of the resource consumption (execution time, disk space, memory) is consumed by 20% of the components.
  - Elaborate the performance-critical components first so that engineering trade-offs with reliability, changeability, and cost effectiveness can be resolved as early in the life cycle as possible.
- 80% of the progress is made by 20% of the people.
  - Make sure the initial team that plans the project and designs the architecture is of the highest quality. An adequate plan and adequate architecture can then succeed with an average construction team. An inadequate plan or inadequate architecture will probably not succeed, even with an expert construction team.

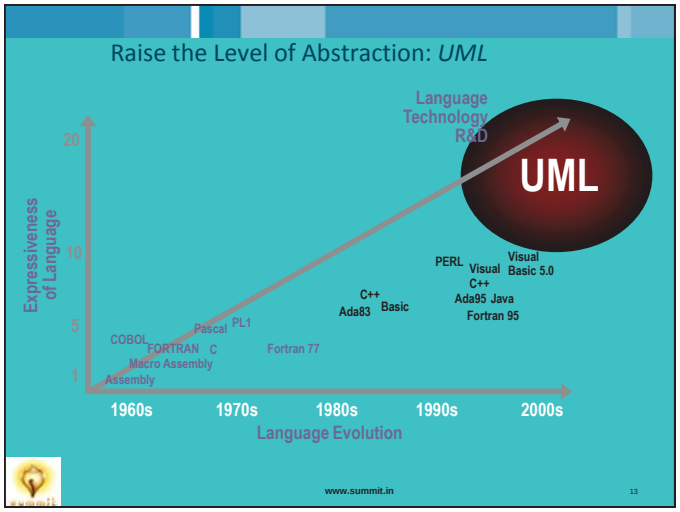
Best Practices – Initial

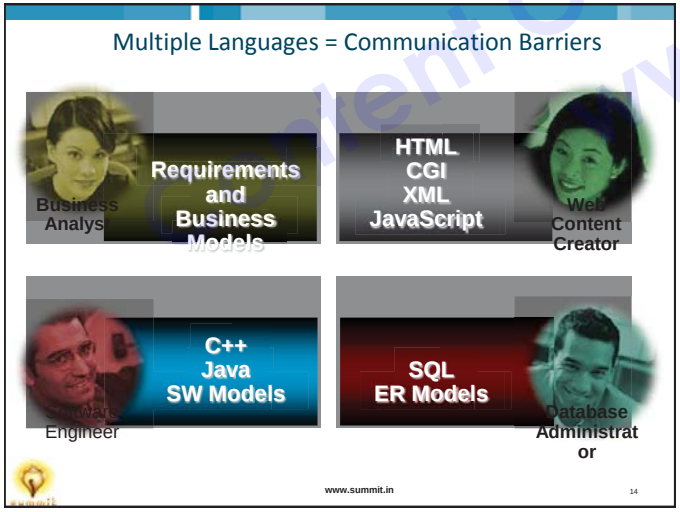
UML

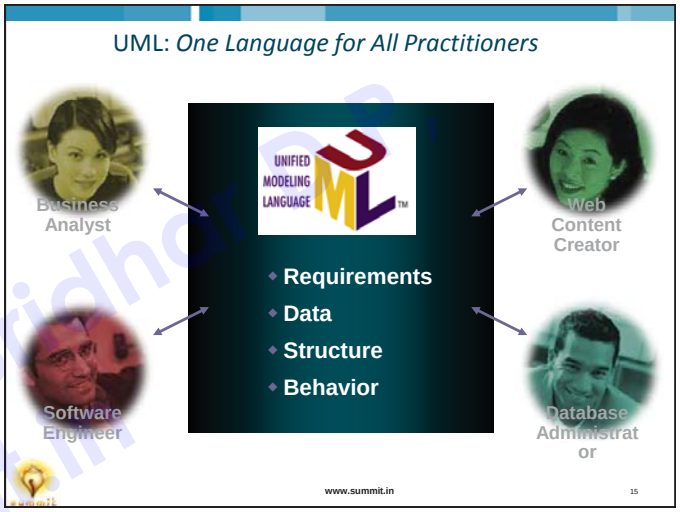
- The UML is a common language for software design
  - Architecting
  - Visualizing
  - Specifying
  - Constructing
  - Documenting

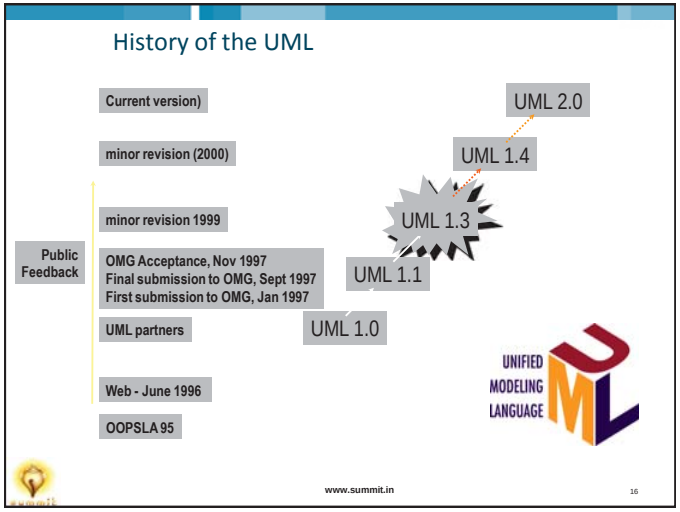
Why Model Visually?

- Capture structure and behavior
- Show how system elements fit together
- Keep design and implementation consistent
- Hide or expose details as appropriate
- Promote unambiguous communication
  - UML provides one language for all practitioners

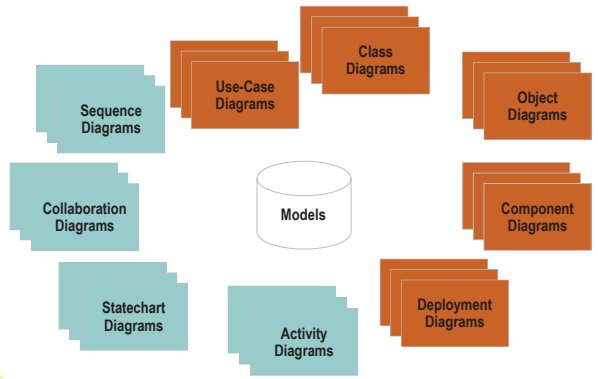








Visual Modeling with Unified Modeling Language



UML 2.0

Structural	Behavioral	Interaction
Class Diagram	Use Case Diagram	Derived from the more general Behavior Diagram
Object Diagram	Activity Diagram	Sequence Diagram
Component Diagram	State Machine Diagram	Communication Diagram
Composite Structure Diagram		Timing Diagram
Package Diagram		Interaction Overview Diagram.
Deployment Diagram		

List of UML diagrams

Requirements
- Use Case diagram
- Activity diagram

Analysis & Design
- Class/Object
- Sequence
- Collaboration also called Communication diagram
- State chart

Implementation
- Component diagram

Physical View
- Deployment

Other UML diagrams
- Package diagrams
- Subsystem diagrams
- Architectural Layer diagram
- Composite Structure diagram

Web links
- www.uml.org
- http://www-306.ibm.com/software/rational/uml/

Disclaimer:

- The UML 2.0 specification is huge
- This presentation is NOT intended to be a comprehensive review of UML 2.0 … it's only a start
- I am focusing on the "end user" perspective of the UML 2.0 changes – NOT the meta-model changes
- I've only highlighted the changes and haven't been able to dive into each change.

UML 2.0 Changes

- ***UML 2.0 - A Major Upgrade:*** In mid-2001, OMG members started work on a major upgrade to UML 2.0.
- Four separate RFPs to keep the effort organized for:
  - *UML Infrastructure*
  - *UML Superstructure*
  - *Object Constraint Language*
  - *UML Diagram Interchange*

UML 2.0 Changes

- **Highlights of the UML 2.0 RFP:**
  - UML 1.x notions of interface and architecture must be enhanced to support and simplify support for standard component frameworks and architectures
  - Data flow modeling must be added
  - Many of the semantics of relationships must be clarified
  - In UML 1.x, sequence diagrams are too limited in their expressiveness and semantics and must be enhanced
  - Activity diagrams should be semantically separated from state machines
  - Clean up inconsistencies and errors in the UML 1.x specifications
  - Superstructure requirements to improve the ability and utility of the UML with the respect to architecture and scalability

UML 2.0 Changes

- The changes for architecture are primarily in the structural (class) model *
- Changes for scalability are best seen in the improved sequence diagrams *
- The UML 2.0 specifications can be found at: http://www.omg.org/uml/
- UML 1.X models will remain valid, OMG strived for backward compatibility
- Officially announced **June 12th, 2003**
- The UML 2.0 specification is expected to be released **April 30, 2004**

* Quoted from I-Logix white paper http://www.ilogix.com/whitepaper_PDFs
UML 2.0: Incremental Improvements for Scalability and Architecture

UML 2.0 Changes

- A quote by Jim Odell, noted consultant and writer as well as the co-chair of the Analysis and Design Task Force:

  "Based on our 5+ years of experience in using UML, we have learned a great deal about unifying modeling languages. Using this knowledge, UML 2.0 literally represents the next evolutionary step in our ability to express and communicate system specifications ---- one which provides a sound basis for MDA"

UML 2.0 Changes

- The upgraded UML standard now has the following features:
  - A first-class extension mechanism allows modelers to add their own metaclasses, making it easier to define new UML Profiles and to extend modeling to new application areas.
  - Built-in support for component-based development to ease modeling of applications realized in Enterprise JavaBeans, CORBA® components or COM+.Support for run-time architectures allows modeling of object and data flow among different parts of a system. Support for executable models improved in general.
  - More accurate and precise representation of relationships improves modeling of inheritance, composition and aggregation, and state machines.
  - Better behavioral modeling improves support for encapsulation and scalability, removes restrictions on mapping of activity graphs to state machines, and improves Sequence diagram structure.
  - Overall improvements to the language simplifies syntax and semantics, and better organizes its overall structure.

UML 2.0 XMI Changes

- In UML 1.x, XMI is a mechanism for exchanging UML models
  - This mechanism did not fully fulfill the goal of model interchange
- The UML 2.0 solution extends the UML metamodel by a supplementary package or graphic-oriented information while leaving the current UML metamodel fully intact.
- See the UML 2.0 Diagram Interchange spec at http://www.omg.org/technology/documents/modeling_spec_catalog.htm for additional information.

UML Terminology

- **UML Description *:**
  A specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems. UML 1.5 incorporates Action Semantics, which adds to UML the syntax and semantics of executable actions and procedures, including their run-time semantics.
- **UML Keywords *:**
  abstraction, action sequence, action state, activity graph, architecture, association, class diagram, collaboration diagram, component diagram, control flow, data flow, deployment diagram, execution, implementation, pins, procedure

* quoted from the OMG website

UML Terminology

- ***What can you Model with UML?***
  - UML defines twelve types of diagrams, divided into three categories:
    - Four diagram types represent static application structure
    - Five represent different aspects of dynamic behavior
    - Three represent ways you can organize and manage your application modules

* quoted from the OMG website

UML Terminology

- **Structural Diagrams** include:
  - *Class Diagram*
  - *Object Diagram*
  - *Component Diagram*
  - *Deployment Diagram*
- **The Class Diagram is the diagram that personally I use on every project, as probably does everyone else.**
- **The other diagrams are used on an as needed basis depending on the size of the project and your role on a project**

UML Terminology

- **Behavior Diagrams** include:
  - *Use Case Diagram*
  - *Sequence Diagram*
  - *Activity Diagram*
  - *Collaboration Diagram*
  - *Statechart Diagram*
- **Sequence Diagrams are generally my Behavior Diagram of choice on projects.**

UML Terminology

- **Model Management Diagrams** include:
  - *Packages*
  - *Subsystems*
  - *Models*

UML 2.0 Highlight of Changes

list is not comprehensive

- Introduced new concept of **Ports**
- **Composite Structure Classes & Diagrams** introduced
- **Class Diagrams** – the least changed
- Collaboration Diagram – renamed to **Communication Diagram**
- **Sequence Diagram –** nesting options
- New diagram introduced **– Timing Diagram**

UML 2.0 Highlight of Changes

list is not comprehensive

- **Activity Diagrams** have the greatest number of changes of any of the UML diagrams
- **Use Case Diagrams** – added multiplicity and changes with extension points.
- **Package Diagram** now an official UML diagram.

UML 2.0 New Terms

- **Port** *
  - A port connects a class's internals to its environment.
  - It functions as an intentional opening in the class's encapsulation through which messages are sent either into or out of the class, depending on the port's provided or required interfaces.
  - A port that has both provided and required interfaces is bidirectional

* Quoted from Morgan Bjorkander & Chis Kobryn's article "Architecting Systems with UML 2.0 in IEEE Software, July/August 2003
http://www.uml-forum.com/out/pubs/IEEE_SW_Jul03_p57.pdf

UML 2.0 New Terms

- **More about Ports** *
  - *Instantiable* connection points.
  - May optionally be used in conjunction with structured classes to allow "part" instances (those inside structured classes) to export out specific services or operations across the enclosing structured class boundary
  - "Paradigm" or "design pattern" rather than technological enhancement
    - Previously done with "interface objects"
  - Explicit connection allows server to identify the client instance
  - Ports support protocol state charts to specify the allowable sets of interactions across the interface
  - Aids in the encapsulation of a component from its environment
  - Ports are based in the CORBA port concept
  - Use is optional

* Quoted from I-Logix white paper http://www.ilogix.com/whitepaper_PDFs
UML 2.0: Incremental Improvements for Scalability and Architecture

UML 2.0 Changes

- **Interface concept has been expanded in two important ways:**
  - UML 1.x interfaces only allow the specification of the offered side, and that notation is kept.
  - UML 2.0 allows you to (optionally) specify the required (client) side as well – depicted with a "socket" type notation.
- Interfaces can have "virtual" attributes
- Interfaces are not instantiable

* Quoted from I-Logix white paper http://www.ilogix.com/whitepaper_PDFs

UML 2.0 New Terms

- **Structured Classes –** a class that is composed of parts with an explicit "nested" notation. The purpose is to model containment hierarchies.
- Example from white paper below: An ElevatorCar is composed of a number of parts – in this case, buttons, a list of destinations, and a door. Similarly, a Floor class has a button to request elevators to go up or down and a position indicator for every elevator that goes to the floor. These are ***structured classes*** because they are broken down into more primitive part objects. In all likelihood, the Door, ElevatorGnome and Shaft classes are also structured classes but their decomposition is shown elsewhere in the model.

* Quoted from I-Logix white paper http://www.ilogix.com/whitepaper_PDFs

UML 2.0 Changes

- **Class diagrams** are the most familiar and popular diagrams in the UML.
  - Not much has changed in this area for UML 2.0.
- Role Names are now called "*Association End Names*" *

* UML 2.0 for Dummies by Michael Jesse Chonoles, James A. Schardt

UML 2.0 Changes

- **Class diagrams** * - there has been a long-standing question about whether to model attributes as text strings within a compartment of the class or as association in the class diagram.
  - An advantage of using associations was that multiplicities could be explicitly shown.
  - But the visibility of attributes was often reserved for the compartmentalized attribute strings.
  - The new version of the UML creates an equivalence relationship between attributes as compartmentalized strings and attributes as associations

* What's New in UML 2.0? December 18, 2003 by Granville Miller
http://community.borland.com/article/images/31881/Together_White_paper__.pdf

What's New in UML 2.0? December 18, 2003 by Granville Miller
http://community.borland.com/article/images/31881/Together_White_paper__.pdf

UML 2.0 Changes

- **Communication (f.k.a. *Collaboration)* Diagrams –** only one change of interest to the diagramming notation.
  - Messages may now be sent concurrently by placing a letter after the sequence number.
  - See whitepaper for diagram example *
  - This was called "*mutually exclusive conditional paths*" in Craig Larman's book "Applying UML and Patterns"

* What's New in UML 2.0? December 18, 2003 by Granville Miller
http://community.borland.com/article/images/31881/Together_White_paper__.pdf

UML 2.0 Changes

- **Sequence Diagrams –** the changes accomplish two primary goals – improve their ability to specify things and to improve their scalability.
  - Can be broken up into "interaction fragments" which may themselves be represented in the same or a separate diagram.

* Quoted from I-Logix white paper http://www.ilogix.com/whitepaper_PDFs

UML 2.0 Changes

- **Sequence diagrams** can be "nested" using operators:
  - sd – named sequence diagram
  - ref – reference to "interaction fragment"
  - loop – repeat interaction fragment
  - alt – selection
  - par – concurrent (parallel) regions
  - seq – partial ordering (default) (aka "weak")
  - strict – strict ordering
  - assert – required (i.e. causal)
  - opt – optional "exemplar"
  - neg – "can't happen" or a negative specification
- The message entry and exit points are called "gates" and allow tools to ensure that the diagrams are compatible and consistent with each other.

* Quoted from I-Logix white paper http://www.ilogix.com/whitepaper_PDFs

UML 2.0 New Terms

- **Timing Diagrams - need more details here!**

UML 2.0 New Terms

- **Activity Diagrams - need more details here!**
  - With UML 2.0, the activity diagram has a mechanism for describing how exceptions are handled – examples in the whitepaper below *.
  - Activity diagrams have been substantially augmented in the UML 2.0 specification.
  - In general, activity diagrams have been a staple of business process modeling and not system modeling

* What's New in UML™ 2? Model Exceptions - by Randy Miller June 30, 2003
http://community.borland.com/article/0,1410,30169,00.html

UML 2.0 New Terms

- **Activity Diagrams ***
  - The intent of these diagrams has changed fairly radically.
  - Activity diagrams not only describe workflow, they also now have some of the features necessary to support the automation of these flows.

* What's New in UML 2.0? December 18, 2003 By Granville Miller
http://community.borland.com/article/images/31881/Together_White_paper_.pdf

UML 2.0 New Terms

- **Use Case Diagrams ***
  - Use Case Multiplicities lie on the association between actors and use cases.
  - The definition of multiplicities in the use case diagram is exactly the same as they are in a class diagram.
  - Extension Points - conditions from UML 2.0 take extension points one step further. They show the actual logic necessary for one use case to extend another. They also show the exact extension point that is used between the two use cases.

* What's New in UML 2? The Use Case Diagram–by Randy Miller June 30, 2003
http://community.borland.com/article/0,1410,30166,00.html

UML 2.0 New Terms

- **Statechart Diagrams**
  - **Statechart inheritance –** need more details here!

UML 2.0 Changes

- **Protocol State Machines –** used to specify operation invocation sequences.
- For example, when landing a plane to be able to specify that the landing gear is lowered before touchdown.
- A protocol state machine is just like a normal state machine except that it is limited – it cannot have entry or exit actions, activities, internal transitions history states and so on. It's purpose is to provide a means to specify allowable sets of operation service in interfaces.

* Quoted from I-Logix white paper http://www.ilogix.com/whitepaper_PDFs

UML 2.0 New Terms

- **Components –** the notation for a component has changed slightly, there is now a component icon in the corner or a stereotype <<component>>.
- The relationship between components and subsystems has been clarified – a subsystem is a kind of component. Subsystems are usually "larger than" components and may contain other components. You can use the <<subsystem>> stereotype to specify a subsystem.

* Quoted from I-Logix white paper http://www.ilogix.com/whitepaper_PDFs

UML 2.0 New Terms

- **Interaction Graphs - need more details here!**

UML 2.0 Books

- **UML 2.0 for Dummies by Michael Jesse Chonoles, James A. Schardt**

  Available at www.amazon.com for $20.99 or $18.95 at www.bookpool.com

- **UML 2.0 Toolkit by by Hans-Erik Eriksson (Author), et al**

  Available at www.amazon.com for $35.00 or $30.50 at www.bookpool.com

UML 2.0 Articles / Websites

- **Artisan Software:** http://www.artisansw.com/pdflibrary/UML_2.0_info.pdf
- **Executable UML: Diagrams for the Future** http://www.devx.com/uml/
- **I-Logix Software** http://www.ilogix.com/whitepaper_PDFs/UML2.0IncrmentalImprovementsforScalabilityandArchitecture.pdf
- **Scott Ambler – Agile Modeling** "The Diagrams of UML 2.0" http://www.agilemodeling.com/essays/umlDiagrams.htm
- **Borland's Developer Community** http://community.borland.com

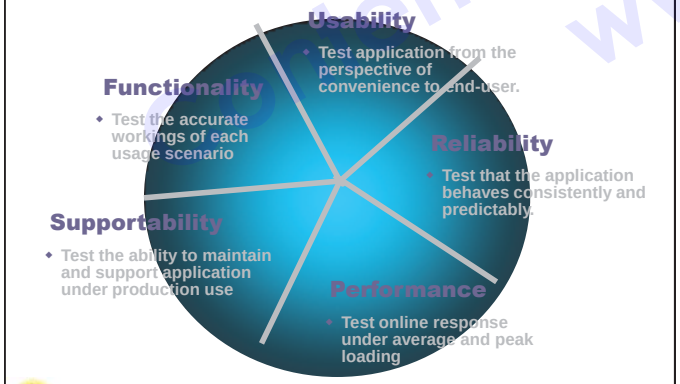
UML Advanced Terminology

- ***Advanced UML Features:*** Two features add to the expressiveness of UML. Object Constraint Language (OCL) has been part of UML since its beginning, while the Action Semantics extension is a recent addition:
  - **Object Constraint Language** lets you express conditions on an invocation in a formally defined way. You can specify invariants, preconditions, postconditions, whether an object reference is allowed to be null, and some other restrictions using OCL. As you might expect, the MDA relies on OCL to add a necessary level of detail to PIMs and PSMs.
  - **Action Semantics UML Extensions** let you express actions as UML objects. An Action object may take a set of inputs and transform it into a set of outputs (although one or both sets may be empty), or may change the state of the system, or both. Actions may be chained, with one Action's outputs being another Action's inputs. Actions are assumed to occur independently - that is, there is infinite concurrency in the system, unless you chain them or specify this in another way. This concurrency model is a natural fit to the distributed execution environment of modern enterprise and Internet applications.

Best Practices Reinforce Each Other

Best Practices	
Develop Iteratively	Ensures users involved as requirements evolve
Manage Requirements	Validates architectural decisions early on
Use Component Architectures	Addresses complexity of design/implementation incrementally
Model Visually (UML)	Measures quality early and often
Continuously Verify Quality	Evolves baselines incrementally
Control Changes (UCM)	

R = F + URPS



Lesson Learnt

- Implement Best Practices
  - Address the root causes
  - Integrate best practices for overall Success
- References
  - Requirements Management – A usecase based approach – Dean Leffingwell.
  - RUP – An Introduction – Dr.Philippe Krutchen.
  - Project Management – A Unified Approach – Walker Royce.
  - COCOMO Model I – Suited Water Fall Model
  - COCOMO II (Iterative Projects)– Barry Boehmn & Walker Royce.

Module 2

Module 2 : Object Thinking

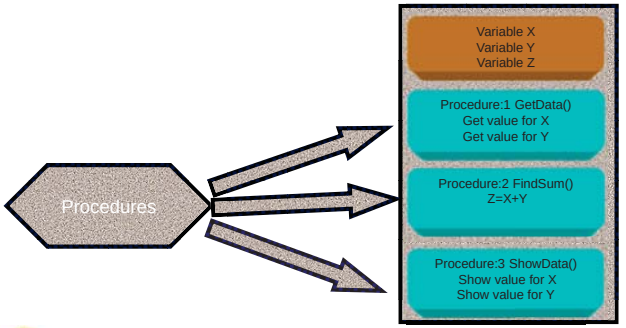
Objectives

- Need for object-oriented programming
  - ✓ About Procedural Programming
  - ✓ Working of Procedural Programming
- Compare OOPS with procedural programming
- Identify the advantages of object-oriented programming
- Identify classes and objects
- Features of OOP

About Procedural Programming Language

- Procedural programming
  - ✓ Involves dividing a large program into a set of subprocedures or subprograms that perform specific tasks.
  - ✓ Module consists of single or multiple procedures.
  - ✓ Procedures are also known as functions, routines, subroutines, or methods in various programming languages.
  - ✓ In a program following procedural methodology, each step of a subprogram is linked to the previous step.

Working of Procedural Programming



Disadvantage of Procedural Programming Language

- **Difficulties of reasoning about programs**
- **Security of data**
- **Procedural programming tends to be relatively low level as a result less productive.**
- **Not suitable for complex real life problems**

About Object Oriented Programming

- Object-Oriented Programming :
  - ✓A large application consists of component objects, which interact with each other.
  - ✓Can be used to develop various applications.

What is object-oriented software development?

- A way to view the world of the application
- A way to describe a model of the application
- A comprehensive methodology that
  - allows to develop a software system
  - uses similar concepts within the whole development process
- Means to achieve high quality
  - Information Hiding
  - Abstraction
  - Modularization
  - Reuse
- An object oriented approach - more or less - forces the software developer to apply these concepts

OO methodologies

- Late 80's early 90: several OO methodologies developed
  - different notations
  - different processes
- Main approaches
  - Booch
  - Rumbaugh
  - Jacobson
  - UML

What is an Object?

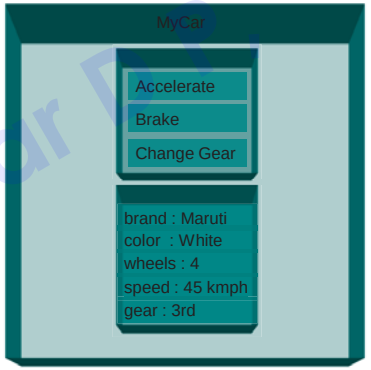
- **An object is a software construct that *encapsulates* data, along with the ability to use or modify that data, into a software entity.**
- **An *object* is a self-contained entity which has its own private collection of *properties* (ie. data) and *methods* (ie. operations) that encapsulate functionality into a reusable and dynamically loaded structure.**

What is an Object (Continued)

- Booch defines an object as:
  "Something you can do things to". An object has:
  - state,
  - behavior, and
  - identity;

  the structure and behavior of similar objects are defined in their common class."

Example



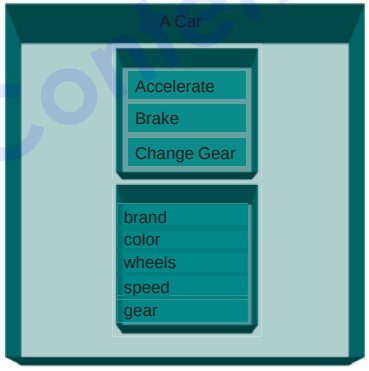
What is an Object Oriented Program

- **An Object-Oriented Program consists of a group of cooperating objects, exchanging messages, for the purpose of achieving a common objective.**

What is a Class?

- **A class is a blueprint or prototype that defines the variables and the methods common to all objects of a certain kind.**
  - blueprint: A class can't do anything on its own.
  - defines: A class provides something that can be used later.
  - objects: A class can only be used, if it had been "brought to life" by instantiating it.

Example

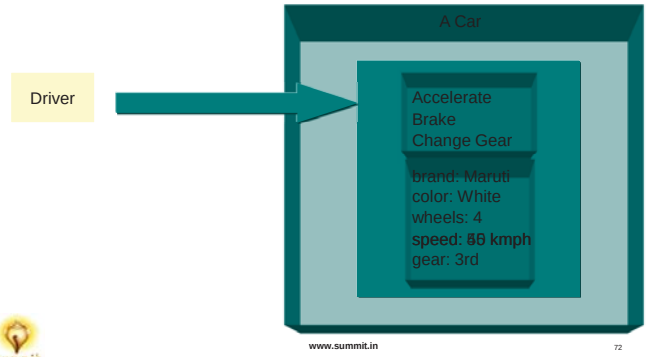


Methods

- **An operation upon an object, defined as part of the declaration of a class.**
- **The methods, defined in a class, indicate, what the instantiated objects are able to do.**

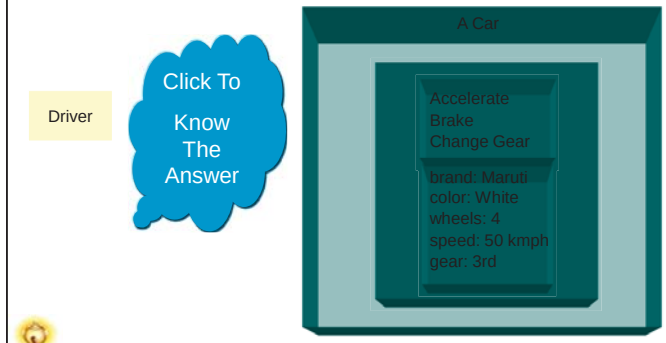
Example

Driver wants to increase the speed of the car?



Question

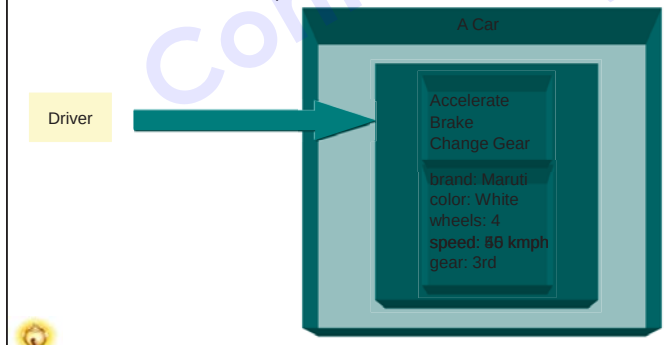
Driver wants to decrease the speed of the car?



www.summit.in 73

Answer

Driver wants to decrease the speed of the car?



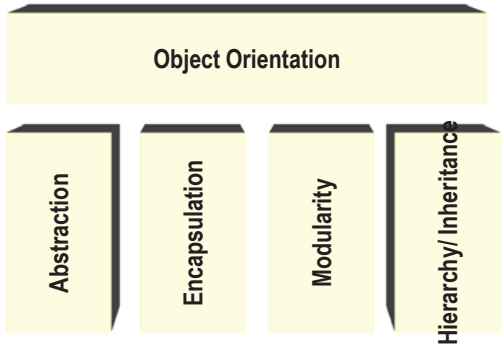
www.summit.in 74

Advantages of OOP

- Real-world programming
- Reusability of code
- Modularity of code
- Resilience to change
- Information hiding

www.summit.in 75

Basic Principles of Object Orientation



www.summit.in 76

Abstraction

"An Abstraction denotes the essential characteristics of an object that distinguishes it from all other kinds of objects and thus provides crisply defined conceptual boundaries, relative to the perspective of the viewer."

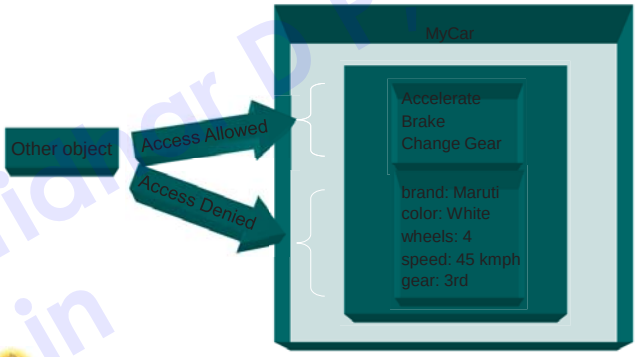
Encapsulation hides the irrelevant details of an object and **Abstraction** makes only the relevant details of an object visible.

Encapsulation

- *Encapsulation* is the ability of an object to place a boundary around its ***properties*** (ie. data) and ***methods*** (ie. operations).
- Grady Booch, defined the encapsulation feature as:

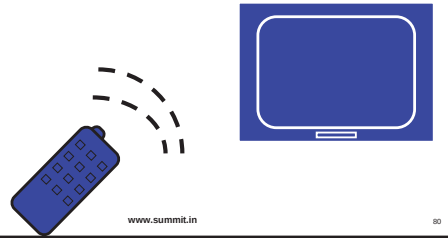
"Encapsulation is the process of hiding all of the details of an object that do not contribute to its essential characteristics."

Encapsulation (Example)



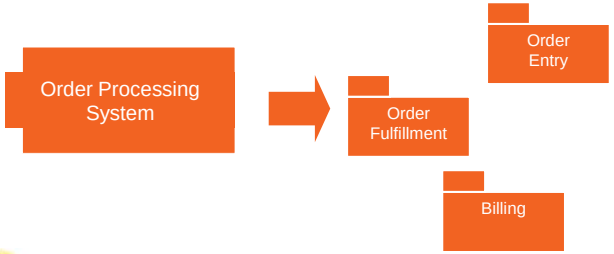
Encapsulation

- **Encapsulation** can be defined as:
- The physical localization of features (e.g., properties, behaviors) into a single blackbox abstraction that hides their implementation (and associated design decisions) behind a public interface. (*Dictionary of Object Technology*, Firesmith, Eykholt, 1995)
- Encapsulation is often referred to as "information hiding," making it possible for the clients to operate without knowing how the implementation fulfills the interface.

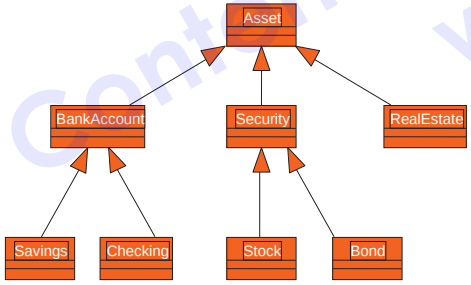


Modularity

- Modularity is the breaking up of something complex into manageable pieces.
- Modularity helps people to understand complex systems.



Hierarchy



- **Hierarchy** can be defined as:
- Any ranking or ordering of abstractions into a tree-like structure. Kinds: Aggregation hierarchy, class hierarchy, containment hierarchy, inheritance hierarchy, partition hierarchy, specialization hierarchy, type hierarchy. (*Dictionary of Object Technology*, Firesmith, Eykholt, 1995)

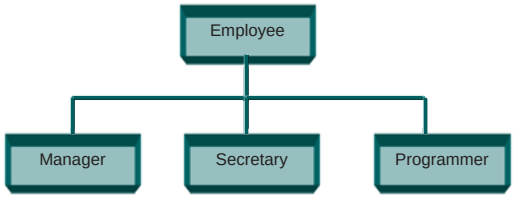
Inheritance

- ***Inheritance*** is the capability of a class to use the properties and methods of another class while adding its own functionality.
- Enables you to add new features and functionality to an existing class without modifying the existing class.

Inheritance (Continued)

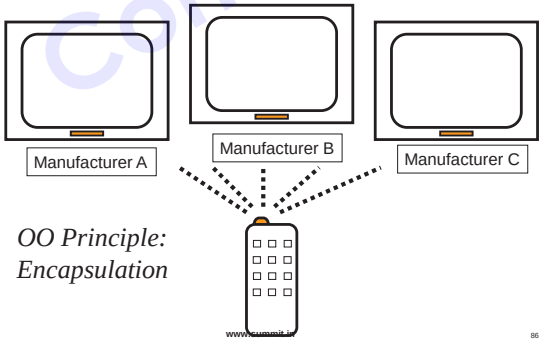
- Superclass and Subclass
  - ✓A *superclass* or *parent* class is the one from which another class inherits attributes and behavior.
  - ✓A *subclass* or *child* class is a class that inherits attributes and behavior from a superclass.

Inheritance (Continued)



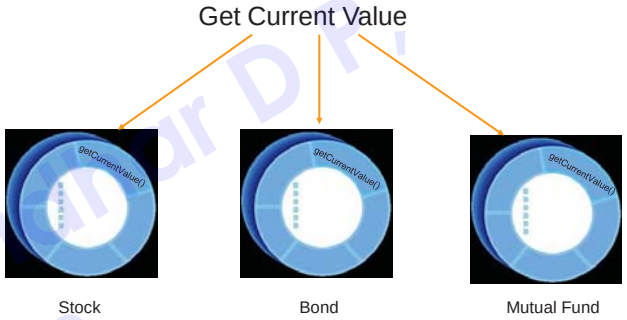
What Is Polymorphism?

- The ability to hide many different implementations behind a single interface



OO Principle: Encapsulation

Example: Polymorphism

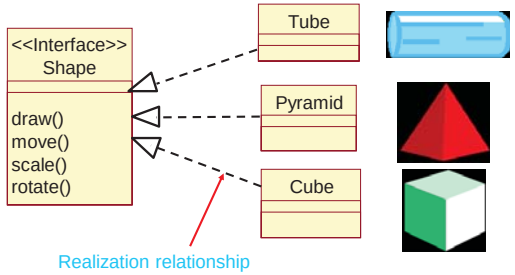


Polymorphism

- Derived from two Latin words - Poly, which means many, and morph, which means forms.
- It is the capability of an action or ***method*** to do different things based on the object that it is acting upon.
- In object-oriented programming, *polymorphism* refers to a programming language's ability to process objects differently depending on their data type or class.

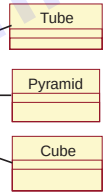
What is an Interface?

- Interfaces formalize polymorphism
- Interfaces support “plug-and-play” architectures

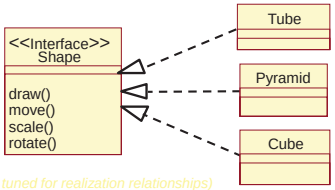


How Do You Represent An Interface?

Elided/Iconic Representation (“lollipop”)



Canonical (Class/Stereotype) Representation



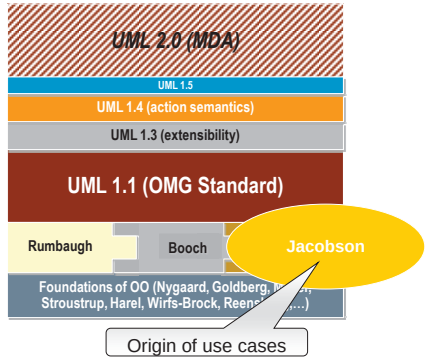
(stay tuned for realization relationships)

UML 2.0

- UML 2 defines 13 basic diagram types, divided into two general sets:
- Structural Modeling diagrams
  - Structure diagrams define the static architecture of a model. They are used to model the 'things' that make up a model - the classes, objects, interfaces and physical components. In addition they are used to model the relationships and dependencies between elements.
- Behavioral Modeling diagrams
  - Behavior diagrams capture the varieties of interaction and instantaneous states within a model as it 'executes' over time; tracking how the system will act in a real-world environment, and observing the effects of an operation or event, including its results.

The Unified Modeling Language (UML)

The Evolution of UML



UML 2.0 – Structural diagrams

- Package diagrams are used to divide the model into logical containers, or 'packages', and describe the interactions between them at a high level.
- Class or Structural diagrams define the basic building blocks of a model: the types, classes and general materials used to construct a full model.
- Object diagrams show how instances of structural elements are related and used at run-time.
- Composite Structure diagrams provide a means of layering an element's structure and focusing on inner detail, construction and relationships.
- Component diagrams are used to model higher level or more complex structures, usually built up from one or more classes, and providing a well defined interface.
- Deployment diagrams show the physical disposition of significant artifacts within a real-world setting.

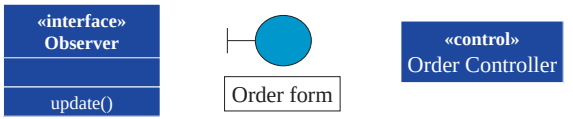
UML 2.0 – Behavioral diagrams

- **- Use Case diagrams are used to model user/system interactions. They define behavior, requirements and constraints in the form of scripts or scenarios.-**
- **Activity diagrams have a wide number of uses, from defining basic program flow, to capturing the decision points and actions within any generalized process.-**
- **State Machine diagrams are essential to understanding the instant to instant condition, or "run state" of a model when it executes.-**
- **Communication diagrams show the network, and sequence, of messages or communications between objects at run-time, during a collaboration instance.-**
- **Sequence diagrams are closely related to communication diagrams and show the sequence of messages passed between objects using a vertical timeline.-**
- **Timing diagrams fuse sequence and state diagrams to provide a view of an object's state over time, and messages which modify that state. - Interaction Overview diagrams fuse activity and sequence diagrams to allow interaction fragments to be easily combined with decision points and flows.**

Some more UML Concepts

Stereotypes

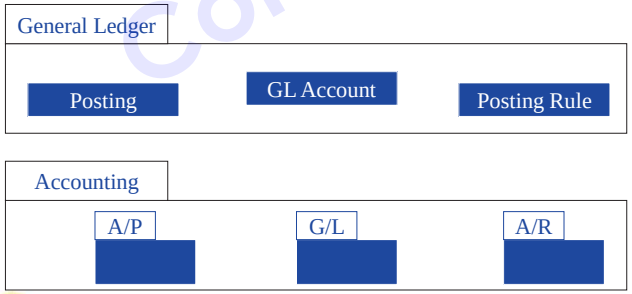
A stereotype is an extension of the vocabulary of the UML that allows you to create a new kind of “building block” that’s specific to the problem you’re trying to solve.



Package

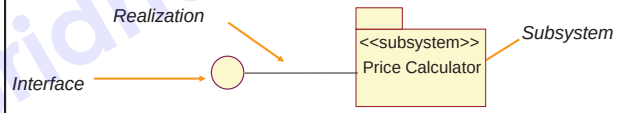
- A package is a general-purpose mechanism for organizing elements of a model, such as classes or diagrams, into groups.
- Every element within a model is uniquely owned by one package. Also, that element's name must be unique within that package.

Sample Package Diagrams



What is a Subsystem?

- A combination of a package (can contain other model elements) and a class (has behavior)
- Realizes one or more interfaces which define its behavior



Component

A component is a physical, replaceable part of a system that conforms to, and provides the realization of, a set of interfaces.

examples:

- dynamic link library (DLL)
- COM+ component
- Enterprise Java Bean (EJB)

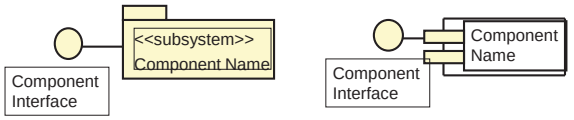
What is a Component?

- A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture
- A component may be
  - A source code component
  - A run time components or
  - An executable component

OO Principle: Encapsulation

Subsystems and Components

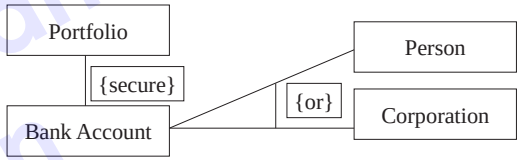
- Components are the physical realization of an abstraction in the design
- Subsystems can be used to represent the component in the design



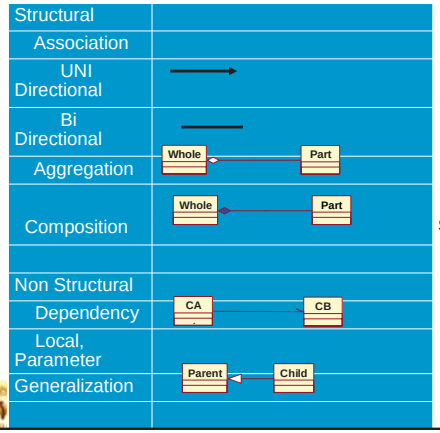
Design Implementation Model

Constraints

A constraint is an extension of the semantics of one or more model elements which specifies a condition that must be true.



Relationships



Independent of each other

Whole Part / has – a / is a part of

Strong Whole Part / Strong has - a

Weaker form of relnship, Using

Is A Kind of

Association

- The semantic relationship between two or more classifiers that specifies connections among their instances
  - A structural relationship, specifying that objects of one thing are connected to objects of another

Multiplicity

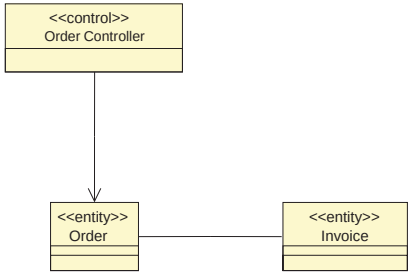
- Multiplicity is the number of instances of one class relates to ONE instance of another class.
- For each association, there are two multiplicity decisions to make, one for each end of the association.
  - For each instance of Order, there can be one or more product.
  - For each instance of Product, there may be either zero or one .

Aggregation

- An aggregation is a special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.
  - An aggregation "Is a part-of" relationship.
- Multiplicity is represented like other associations.

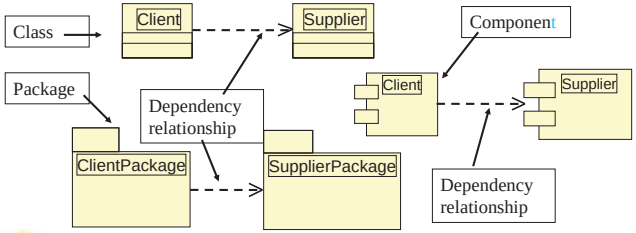
Navigability

- Indicates that it is possible to navigate from a associating class to the target class using the association



Relationships: Dependency

- A relationship between two model elements where a change in one may cause a change in the other
- Non-structural, “using” relationship

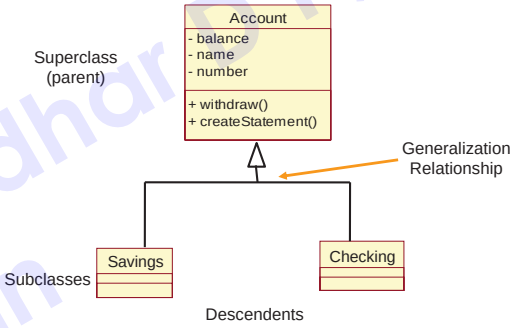


What Is Generalization?

- A relationship among classes where one class shares the structure and/or behavior of one or more classes
- Defines a hierarchy of abstractions in which a subclass inherits from one or more superclasses
  - Single inheritance
  - Multiple inheritance
- Is an “is a kind of” relationship

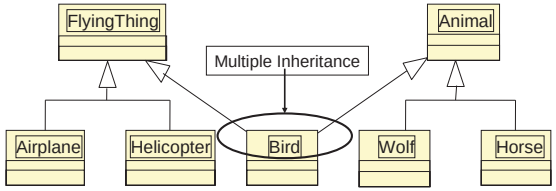
Example: Single Inheritance

- One class inherits from another



Example: Multiple Inheritance

- A class can inherit from several other classes.



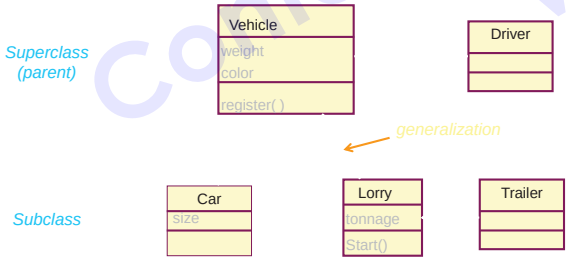
Use multiple inheritance only when needed

What Gets Inherited?

- A subclass inherits its parent's attributes, operations, and relationships
- A subclass may:
  - Add additional attributes, operations, relationships
  - Redefine inherited operations (use caution!)
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy

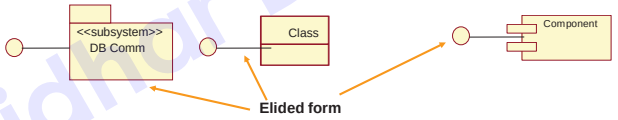
Inheritance leverages the similarities among classes

Example: What Gets Inherited



What Is Realization?

- One classifier serves as the contract that the other classifier agrees to carry out
- Found between:
  - Interfaces and the classifiers that realize them



KnowledgeWorks
REFINING THOUGHTS

summit
Freedom To Freedom

Module 2 : UML in Requirements Phase

Consulting . Innovation. Mentoring

www.summit.in

1

What is a requirement?

A requirement describes a condition or capability to which a system must conform; either derived directly from user needs, or stated in a contract, standard, specification, or other formally imposed document.*

***Rational Unified Process**

Requirements are all about making sure that you build the right thing

www.summit.in

2

What is requirements management?

Ensuring that your team identifies, builds, tests and documents the right system for your customer

A systematic approach to eliciting, documenting, organizing, and tracking changing requirements.

Requirements represent a contract between your customer and your team

www.summit.in

3

Who needs requirements?

All project team members need access to requirements

www.summit.in

4

Define the system: requirements types

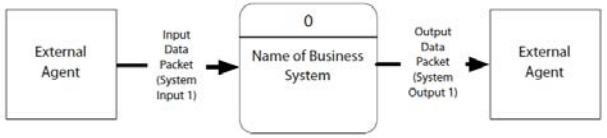
Classify requirements by type

FURPS	Legal and Regulatory	Design Constraints
Functionality Usability Reliability Performance Supportability	▶ FCC ▶ FDA ▶ DOD ▶ ISO	▶ Operating systems ▶ Environments ▶ Compatibility ▶ Application standards ▶ System integrations

www.summit.in 5

Context Diagram

- Top-level view of IS
- Shows the system boundaries, external entities that interact with the system, and major information flows between entities and the system.



www.summit.in 6

What is a Use Case?

A use case describes a sequence of actions a system performs that yields an observable result of value to a particular actor

- Use cases are shown in UML diagrams

- An actor represents anything that interacts with the system.
- Use cases are described in text
  - They tell the story of the interactions between actors and the system

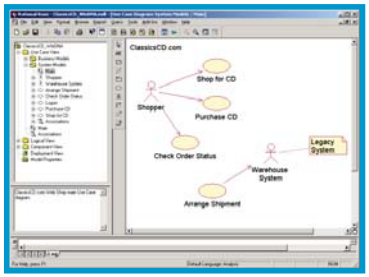
www.summit.in 7

Use Cases vs. Declarative Statements

Declarative	Use Cases
▪ The system shall accept insertion of ATM cards ▪ The system shall offer options for issuing receipts ▪ The system shall require valid PIN#'s ▪ The system shall support deposits, withdrawals and balance inquiries.	1. The user inserts their card 2. The system asks for a identification and the user enters it 3. The system presents options for deposit, withdrawal or balance inquiry. The user choose withdrawal 4. … 5. The user indicates that she/he would like a printed receipt... 6. The use case ends
▶ "The system shall…" ▶ Small perspective ▶ System orientation	▶ Broad perspective ▶ Goal orientation ▶ Actor (user) focus

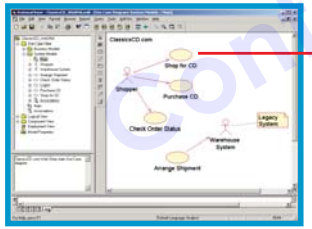
www.summit.in 8

Use cases are graphical…

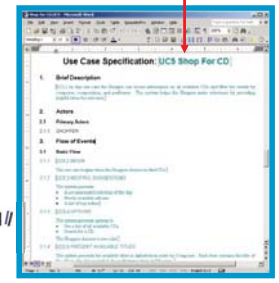


Identified graphically in Modeling tools

Use cases are graphical…but are mostly textual



Identified graphically in Modeling tools



Described textually in Word / Requisite Pro

Usecase Refined Template

- Name
- Brief Description
- Basic Flow
- Alternative Flow
- Pre Condition
- Post Condition
- Extension Points
- Data Points
- Other Diagrams

Benefits of use cases

- Facilitate efficient communication between end users and customers, and the development team
  - Provide context around requirements by expressing sequences of events
  - Use case diagrams act as a 'big picture' of the system
- Defines what the system does to satisfy its stakeholders
- Help reduce design constraints
  - Focus on the "what" not the "how"
- Are reusable by the rest of the team
  - For design, usability design and testing

Who Reads Use Cases?

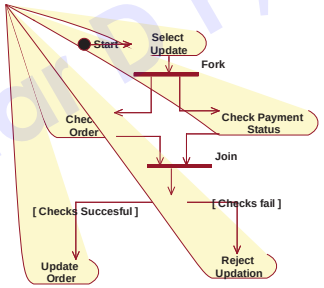
- Client team

- Developer team

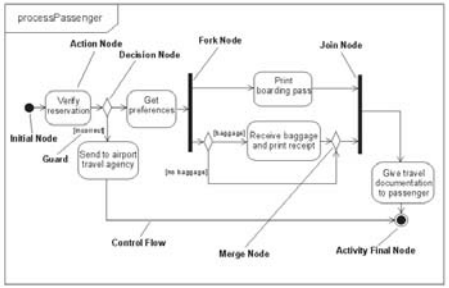
The Role of Activity diagram (P P only)

- Flow Chart - Sequential steps
- Activity diagram – Flowchart + Concurrent activities
  - To explain a use case in a diagrammatic way.
- Swim Lane diagram – Activity diagram + Roles
  - To represent business process, work flow etc.,

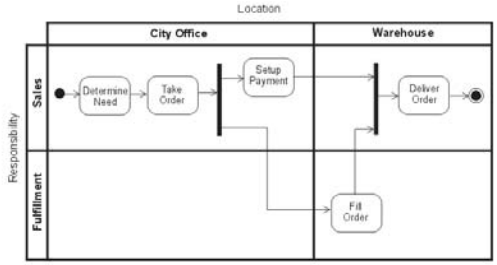
Activity Diagram



Process Passenger- Activity diagram



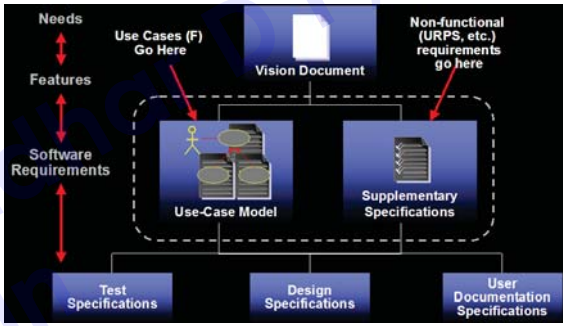
Swimlane diagram



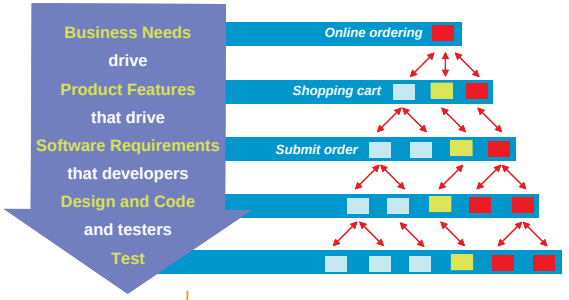
Functional Page Flow(PP)

- Don't Include too much of Gui Design
- Paper Sketches
- GUI Story boards

Define the system: requirements documents



Establish relationships



Check Points

- Lab Exercise
- Check points

Module 3 : UML in Analysis Phase

Consulting . Innovation. Mentoring

Analysis Versus Design

- Analysis
  - Focus on understanding the problem
  - Idealized design
  - Behavior
  - System structure
  - Functional requirements
  - A small model
- Design
  - Focus on understanding the solution
  - Operations and Attributes
  - Performance
  - Close to real code
  - Object lifecycles
  - Non-functional requirements
  - A large model

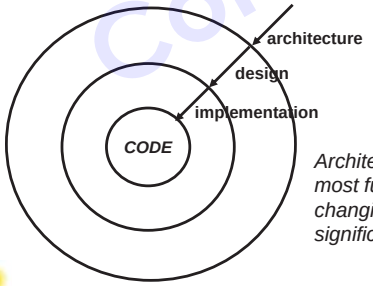
Architecture defined

- IEEE 1471-2000
  - Software architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution
- Software architecture encompasses the set of significant decisions about the organization of a software system
  - Selection of the structural elements and their interfaces by which a system is composed
  - Behavior as specified in collaborations among those elements
  - Composition of these structural and behavioral elements into larger subsystems
  - Architectural style that guides this organization

Booch, Kruchten, Reitman, Bittner, and Shaw

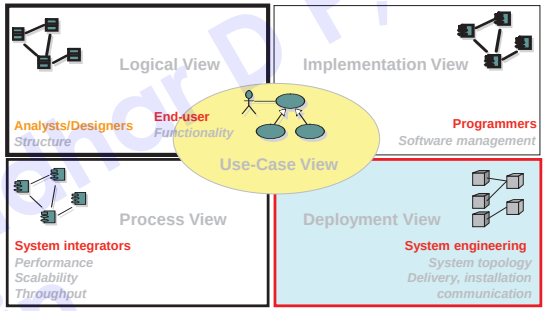
Architecture defined

- Architecture establishes the context for design and implementation

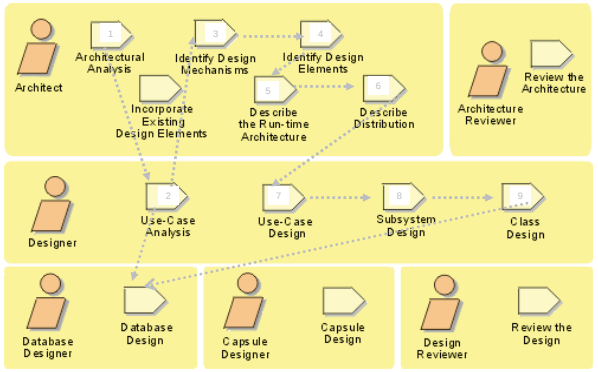


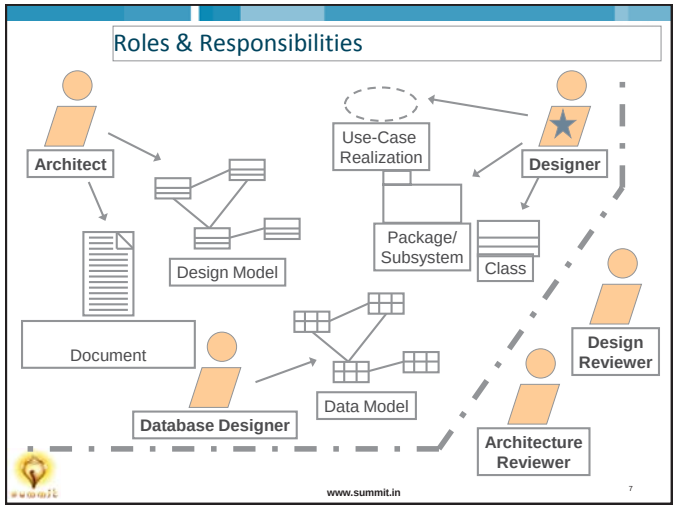
Architectural decisions are the most fundamental decisions; changing them will have significant ripple effects.

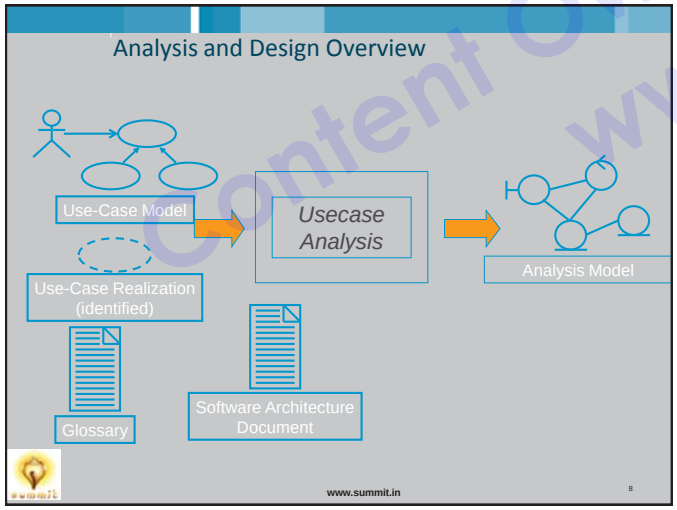
Key Concepts: 4+ 1 View

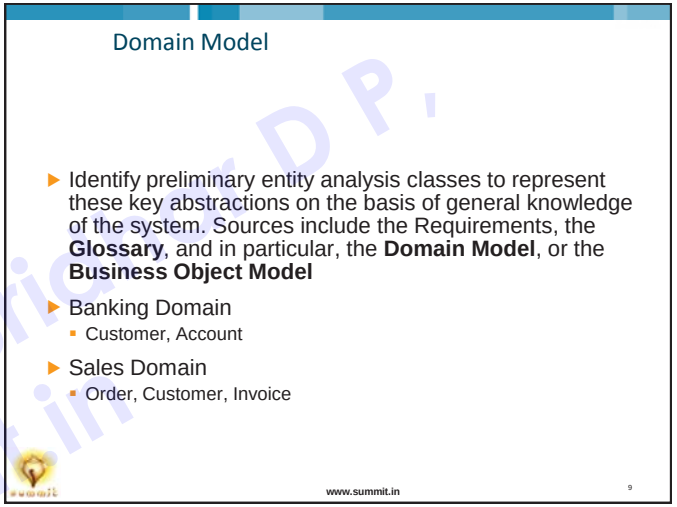


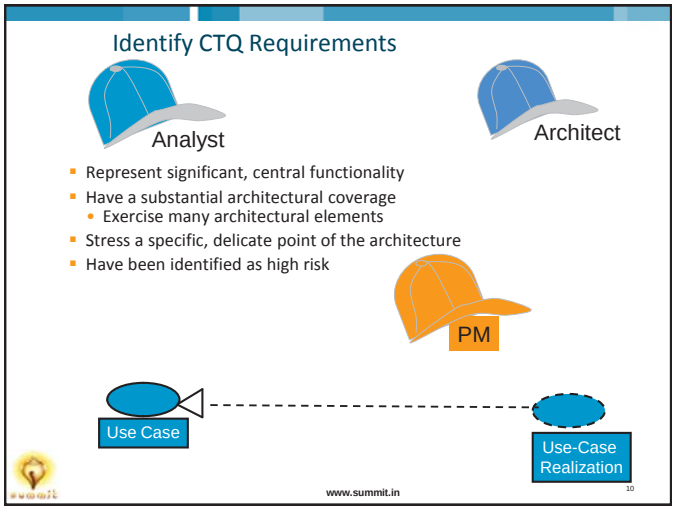
Analysis and Design Activity Overview







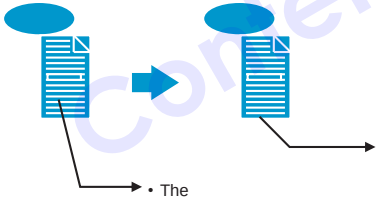




Activities

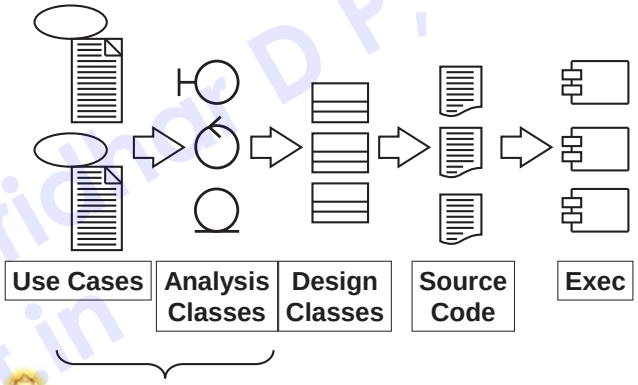
- Check Usecase Specification
- Identify Analysis Classes for a Usecase Specification
- Describe Responsibilities
- Describe Relationships
- Describe Attributes & Multiplicities

Check Usecase Specification



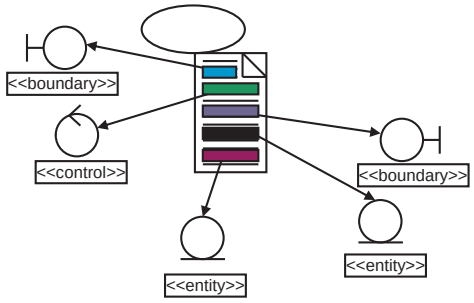
- The system displays a list of Products.
- The system retrieves and displays a list of latest products from Inventory DB.

A First Step Towards Executables

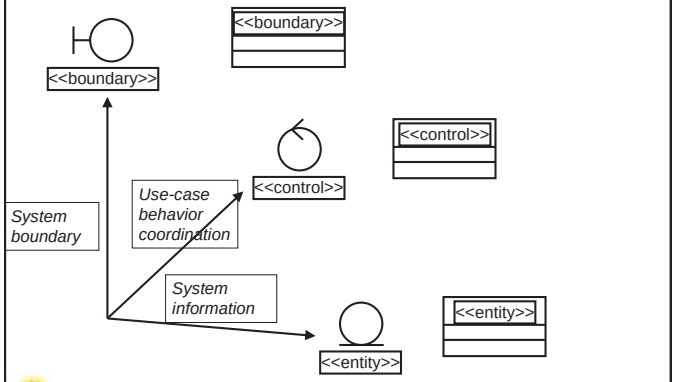


Find Classes From Use-Case Behavior

- The complete behavior of a Use-Case has to be distributed to analysis classes.



What is an Analysis Class?

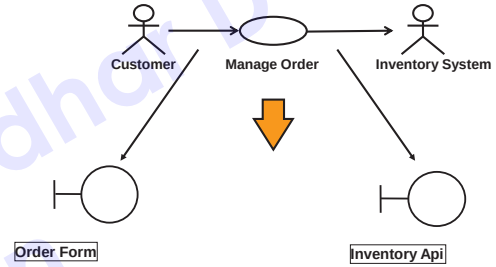


Boundary Classes

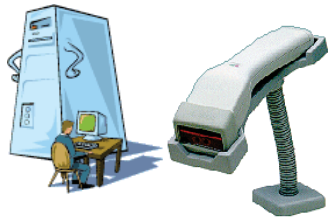
- acts as intermediate between something outside the system (Actor) and the system under development
  - User interface classes
  - System interface classes
  - Device interface classes
- *One boundary class per actor/use case pair*

Environment dependent

Example



Examples

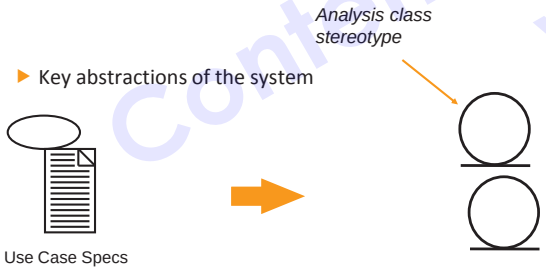


Take Care !

- Don't get into GUI design
  - There can be more than one forms in your design
- Don't ask implementation questions if it is SIBC or DIBC

What is an Entity Class?

- Key abstractions of the system

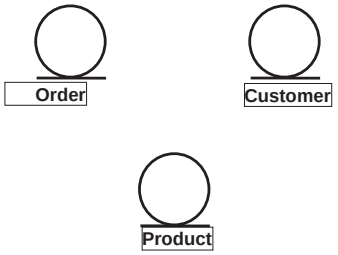


Environment Independent

Entity Identification

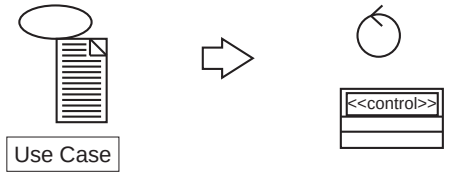
- Traditional, filtering nouns approach
  - Underline noun clauses in the use-case flow of events
  - Remove redundant candidates
  - Remove vague candidates
  - Remove actors (out of scope)
  - Remove implementation constructs
  - Remove attributes (save for later)
  - Remove operations

Entity Classes



What is a Control Class?

- Use case behavior coordinator
- One control class per use case

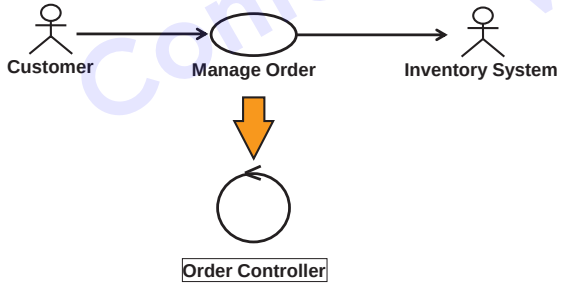


Use case dependent, Environment independent

www.summit.in 23

Example: Finding Control Classes

- One control class per use case

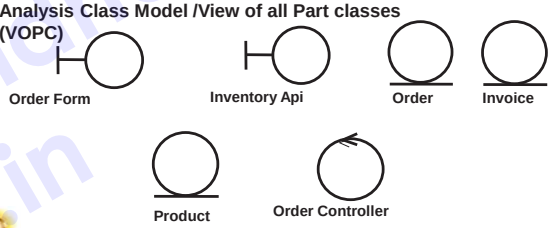


www.summit.in 24

Example: Summary: Analysis Classes

Use-Case Model

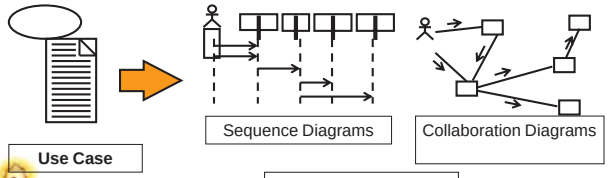
Analysis Class Model /View of all Part classes (VOPC)



www.summit.in 25

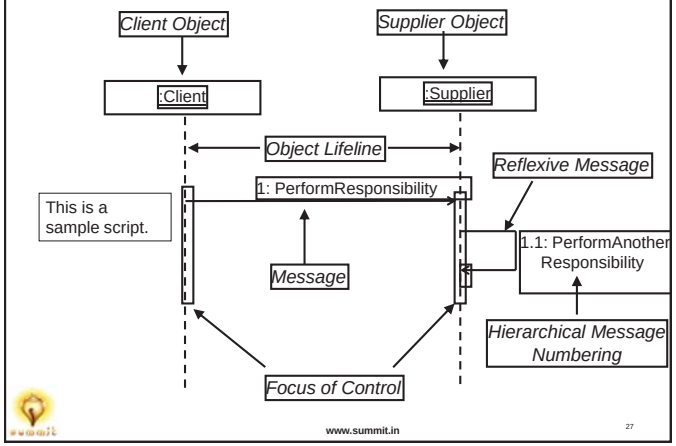
Distribute Use-Case Behavior to Classes

- For each use-case flow of events:
  - Identify analysis classes
  - Allocate use-case responsibilities to analysis classes
  - Model analysis class interactions in interaction diagrams



www.summit.in 26

The Anatomy of Sequence Diagrams

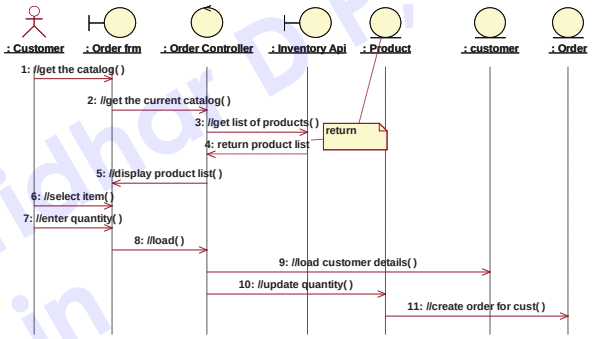


CRC cards (PP Only)

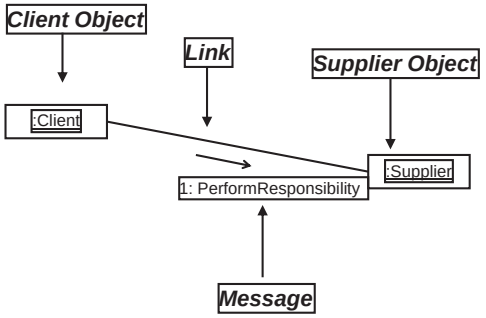
- Class Responsibility Collaborator
- Class Automobile

Responsibilities	Collaborator
Starts itself	
Stops itself	
Change Tire	Mechanic, Tire
Drive	Driver
Get washed	Car Wash, Attendant

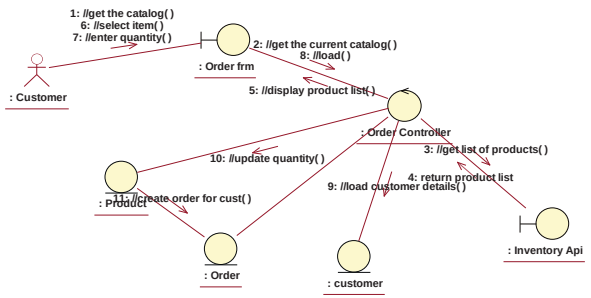
Sequence Diagram



The Anatomy of Collaboration Diagrams



Collaboration diagram



Collaboration Diagrams Vs Sequence Diagrams

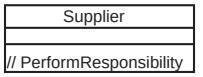
- Collaboration Diagrams
  - Show relationships in addition to interactions
  - Better for visualizing patterns of collaboration
  - Better for visualizing all of the effects on a given object
  - Easier to use for brainstorming sessions
- Sequence Diagrams
  - Show the explicit sequence of messages
  - Better for visualizing overall flow
  - Better for real-time specifications and for complex scenarios

Describe Responsibilities

- What are responsibilities?
- How do I find them?

Interaction Diagram

Class Diagram



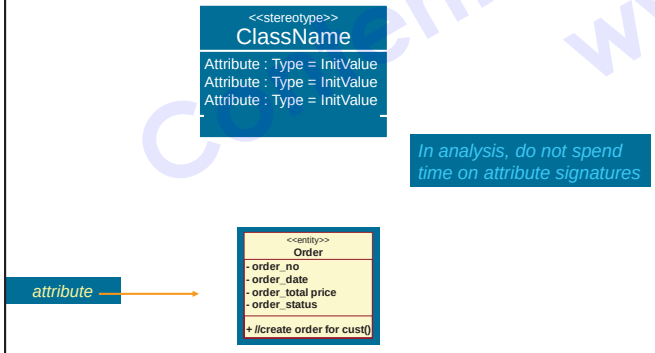
Maintaining Consistency: What to Look For

- In order of criticality
  - Redundant responsibilities across classes
  - Disjoint responsibilities within classes
  - Class with one responsibility
  - Class with no responsibilities
  - Better distribution of behavior
  - Class that interacts with many other classes

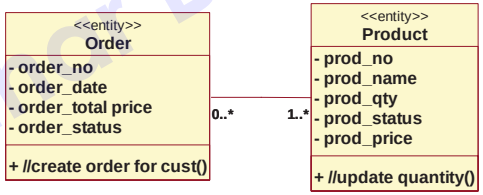
Finding Attributes

- Properties/characteristics of identified classes
- Information retained by identified classes
- "Nouns" that did not become classes
  - Information whose value is the important thing
  - Information that is uniquely "owned" by an object
  - Information that has no behavior

Review: What is an Attribute?



Multiplicity



Example: Describing Analysis Mechanisms

- Analysis class to analysis mechanism map

Analysis Class	Analysis Mechanism(s)
Order	Persistency, Security
invoice	Persistency, Security
Product	Persistency, Legacy Interface
item	Persistency, Legacy Interface
Order Controller	Distribution

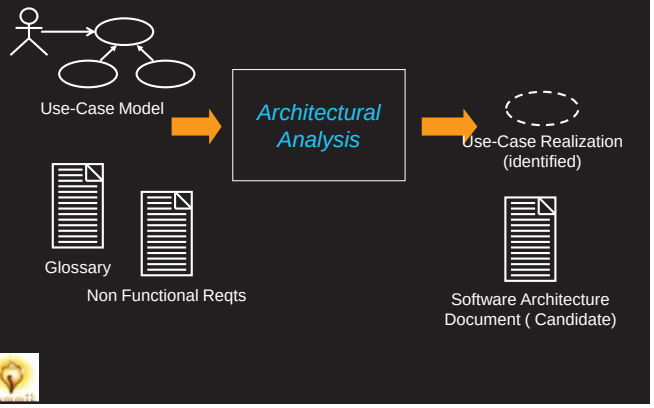
Check Points

- Lab Exercise
- Check points

Module 4 : UML in Architecture definition Phase

Consulting . Innovation. Mentoring

Architectural Analysis



Purpose

- To define a candidate architecture for the system based on experience gained from similar systems or in similar problem domains.
- To define the architectural patterns, key mechanisms, and modeling conventions for the system.
  - To facilitate system envisioning by exploring and evaluating high-level architectural options.
  - To convey an early understanding of the high-level structure of the intended system to the sponsor, development teams, and other stakeholders.

Activities

- Identify Arch.Patterns & Frameworks
- Identify Mechanisms
- Domain Model
- CTQ Requirement & Risk Mitigation
- References
  - Pattern Oriented Software Architecture (POSA) – F.Buschmann & M stale
  - Software Architecture Perspectives – David Garlan & Mary Shaw

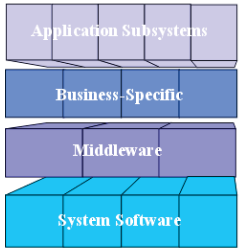
Patterns

- A **pattern** codifies specific knowledge collected from experience. Patterns provide examples of how good modeling solves real problems, whether you come up with it yourself or you reuse someone else's.
  - Ex: Business Pattern , Architectural Pattern, Design Pattern
- **Frameworks** differ from analysis and design patterns in their scale and scope. Frameworks describe a skeletal solution to a particular problem which may lack many of the details, which may be filled in by applying various analysis and design patterns.

Architectural Pattern

Category	Pattern
Structure	**Layers** **Pipes and Filters** **Blackboard**
Distributed Systems	**Broker**
Interactive Systems	**Model-View-Controller** **Presentation-Abstraction-Control**
Adaptable Systems	**Reflection** **Microkernel**

Layer



Distinct Application subsystems that make up an application - contain the value adding software developed by the organization.

Business Specific- contains a number of reusable subsystems specific to the type of business.

Middleware - offers subsystems for utility classes and platform-independent services for distinguished object computing in hetero-geneous environments and so on.

System software - contains the software for the actual infrastructure such as operating systems, interfaces to specific hardware, device drivers and so on.

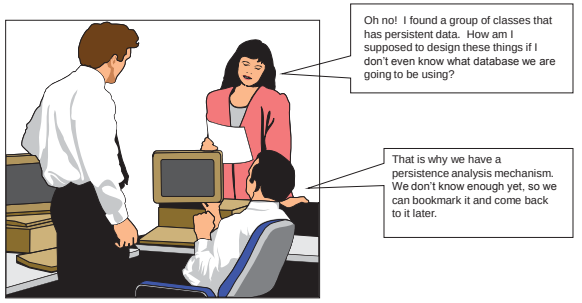
Frameworks

- Business Frameworks
  - Telecom – e Tom
- Architectural Frameworks
  - J2EE
  - .Net
- Product Frameworks
  - SAP , Oracle Financials

Mechanism

- Mechanisms allow the analysis effort to focus on translating the functional requirements into software concepts without bogging-down in the specification of relatively complex behavior needed to support the functionality but not central to it.

Why Use Analysis Mechanisms?



Analysis mechanisms are used during analysis to reduce the complexity of analysis, and to improve its consistency by providing designers with a short-hand representation for complex behavior.

Examples

Analysis Mechanisms
Persistency
Granularity
Volume
Retrieval mechanism
Update Frequency
Reliability
Duration
Inter -Process Communication
Latency
Synchronicity
Size of Message
Protocol

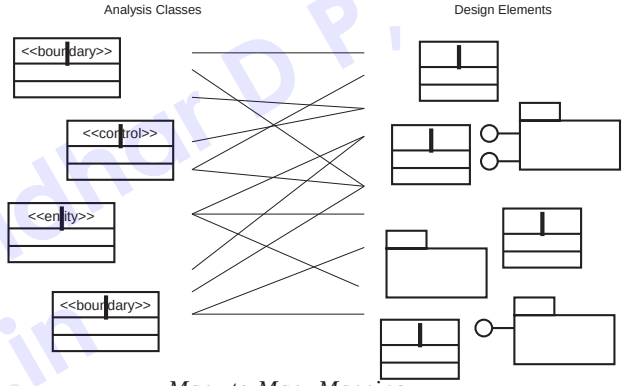
Security & Authentication
User Granularity
Data Granularity
Algorithmn

Sample Mechanisms
Message routing
Process control and synchronization
Transaction management
Information Exchange
Security
Redundancy
Error reporting
Format conversion

Activities

- Packages
- Subsystems
- Logical Architecture

From Analysis Classes to Design Elements



Many-to-Many Mapping

Analysis to Design Element

- An analysis class can become a single design class in the design model.
- An analysis class can become a part of a design class in the design model.
- An analysis class can become an aggregate design class in the design model.
- An analysis class can become a group of design classes that inherits from the same class in the design model.
- An analysis class can become a group of functionally related design classes in the design model.
- An analysis class can become a design subsystem in the design model.

Analysis to Design Element (Cont'd)

- An analysis class can become part of a design subsystem, such as one or more interfaces and their corresponding implementation.
- An analysis class can become a relationship in the design model.
- A relationship between analysis classes can become a design class in the design model.
- Analysis classes handle primarily functional requirements, and model objects from the "problem" domain; design classes handle non-functional requirements, and model objects from the "solution" domain.
- Analysis classes can be used to represent "the objects we want the system to support," without taking a decision on how much of them to support with hardware and how much with software. Thus, part of an analysis class can be realized by hardware, and not modeled in the design model at all.

www.summit.in 15

Package

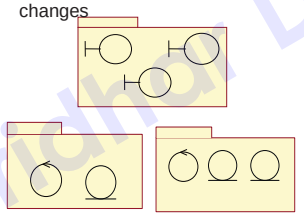
- You can base your packaging criteria on a number of different factors
  - Configuration units
  - Allocation of resources among development teams
  - Reflect the user types
  - Represent the existing products

  and services the system uses
- Design packages are used to group related Design Model elements together for organizational purposes, and often for configuration management
- Package Guidelines

www.summit.in 16

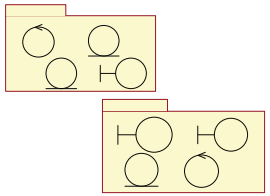
Packaging Tips: Boundary Classes

If it is likely the system interface will undergo considerable changes



Boundary classes placed in separate packages

If it is unlikely the system interface will undergo considerable changes



Boundary classes packaged With functionally related classes

www.summit.in 17

Packaging Tips: Functionally Related Classes

- Criteria for determining if classes are functionally related
  - Changes in one class' behavior and/or structure necessitate changes in another class
  - Removal of one class impacts the other class
  - Two objects interact with a large number of messages or have a complex intercommunication
  - A boundary class can be functionally related to a particular entity class if the function of the boundary class is to present the entity class
  - Two classes interact with, or are affected by changes in the same actor

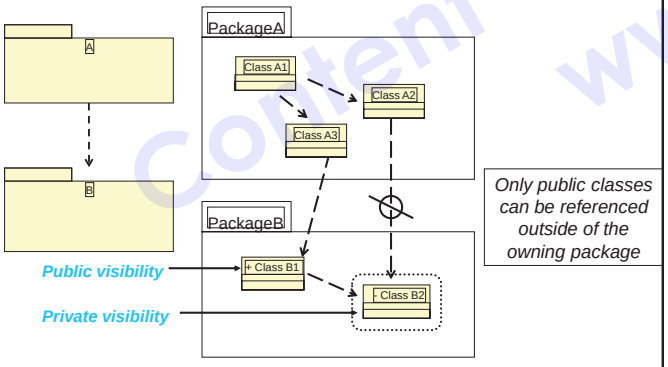
Continued…

www.summit.in 18

Packaging Tips: Functionally Related Classes (cont.)

- Criteria for determining if classes are functionally related (continued)
  - Two classes have relationships between each other
  - One class creates instances of another class
- Criteria for determining when two classes should *NOT* be placed in the same package
  - Two classes that are related to different actors should not be placed in the same package
  - An optional and a mandatory class should not be placed in the same package

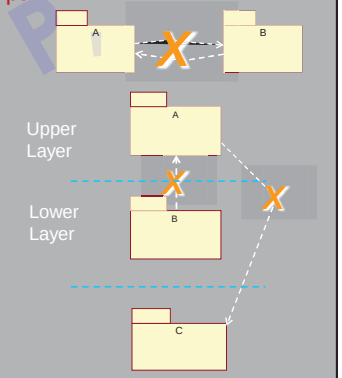
Package Dependencies: Package Element Visibility



OO Principle: Encapsulation

Package Coupling: Tips

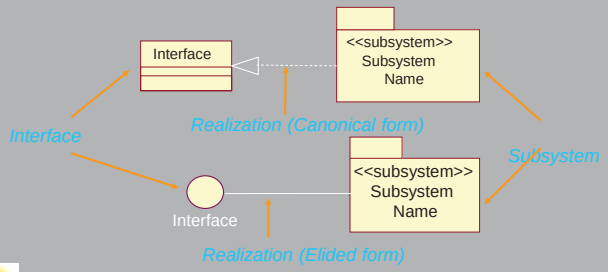
- Packages should not be cross-coupled
- Packages in lower layers should not be dependent upon packages in upper layers
- In general, dependencies should not skip layers



X = Coupling violation

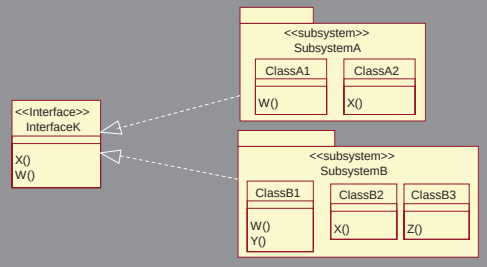
Review: Subsystems and Interfaces

- A "cross between" a package (can contain other model elements) and a class (has behavior)
- Realizes one or more interfaces which define its behavior



Subsystems and Interfaces (cont.)

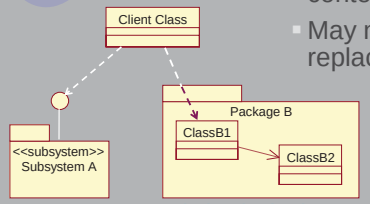
- Subsystems :
  - Completely encapsulate behavior
  - Represent an independent capability with clear interfaces (potential for reuse)
  - Model multiple implementation variants



Packages versus Subsystems

Subsystems
- Provide behavior
- Completely encapsulate their contents
- Are easily replaced

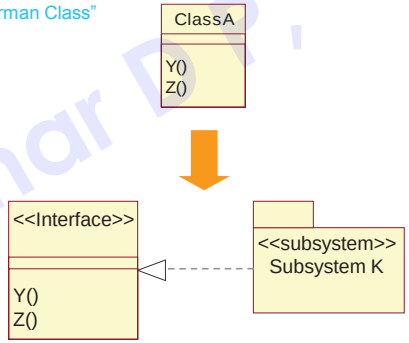
Packages
- Don't provide behavior
- Don't completely encapsulate their contents
- May not be easily replaced



Encapsulation is the key!

Identifying Subsystems

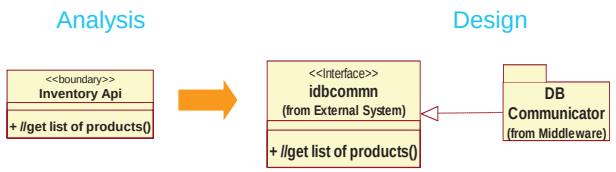
"Superman Class"



Identifying Subsystem hints

- Identifying Subsystems Hints
- Look at object collaborations
- Look for optionality
- Look to the user interface of the system
- Look to the Actors
- Look for coupling and cohesion between classes
- Look at substitution
- Look at distribution
- Look at volatility

Example: Design Subsystems and Interfaces

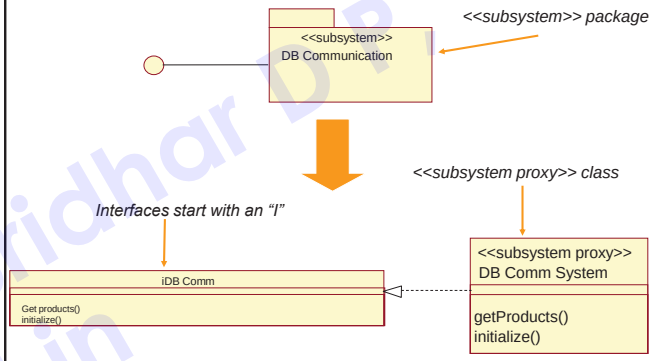


All other analysis classes map directly to design classes

Example: Analysis-Class-To-Design-Element Map

Analysis Class	Design Element
Inventory Api	DB Communication System
Order	Order

Modeling Convention: Subsystems and Interfaces



Identification of Reuse Opportunities

- Purpose
  - To identify where existing subsystems and/or components may be reused based on their interfaces.
- Steps
  - Look for similar interfaces
  - Modify new interfaces to improve the fit
  - Replace candidate interfaces with existing interfaces
  - Map the candidate subsystem to existing components

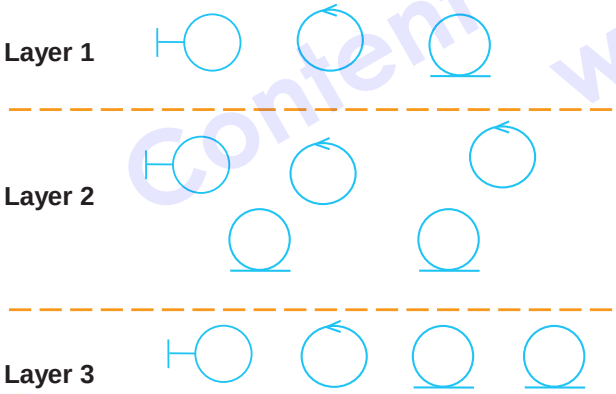
Layering Considerations

- Visibility
  - Dependencies only within current layer and below
- Volatility
  - Upper layers affected by requirements changes
  - Lower layers affected by environment changes
- Generality
  - More abstract model elements in lower layers
- Number of layers
  - Small system: 3-4 layers
  - Complex system: 5-7 layers

Goal is to reduce coupling and to ease maintenance effort

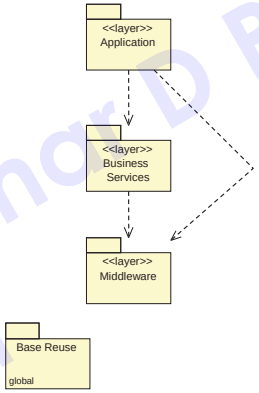
www.summit.in 31

Design Elements and the Architecture



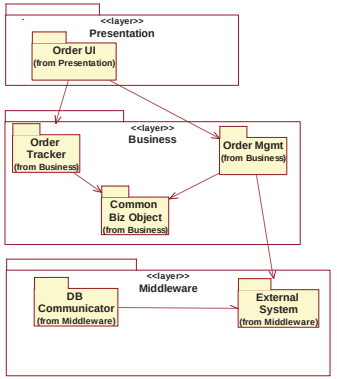
www.summit.in 32

Example: Architectural Layers



www.summit.in 33

Example Order Management System



www.summit.in 34

Design & Implementation Mechanisms

- **Identify the clients of each analysis mechanism.**
  - Scan all clients of a given analysis mechanism, looking at the characteristics they require for that mechanism. For example, a number of Analysis Classes may make use of a Persistence mechanism
- **Identify characteristic profiles for each analysis mechanism.**
  - There may be widely varying **characteristics profiles**, providing varying degrees of performance, footprint, security, economic cost, etc. Each analysis mechanism is different - different characteristics will apply to each. Many mechanisms will require estimates of the number of instances to be managed, and their expected size in terms of the expected number of bytes.
- **Group clients according to their use of characteristic profiles.**
  - Form groups of clients that seem to share a need for an analysis mechanism with a similar characteristics profile; identify a design mechanism based on each such need.

www.summit.in 35

AAM						
	CLASSES			*DM*	*IM*	Local Patterns / Mechanisms
Persistency	**Order, Customer**	**Order Cfm**			Versant	
Granularity	2kb ; 10 kb	1 kb	G1	**OODBMS**	Object store	OO to OODBMS
Volume	100,000/yr ; 10,000/yr				Jasmine	
Acc Freq	C-2mins; u- 3mins; D- 3mins	10 msecs				
Acc Mech	**NA**					
Security	Order, Invoice		G3	Caching	XML Store	
Data Granularity						
User Granularity					128 bit SSL	Security Pattern
Security Rules						
Legacy Access	Product		G2	RDBMS (Known;design constraint)	Oracle	OR Mapper
Latency	30 Secs					VO- J2EE
Acc Mechanism				**MTS-RDBMS**		
Distribution	Order Controller				RMI, EJB	Distributrion pattern
Protocol						
Synchronicity						

www.summit.in 36

Check Points

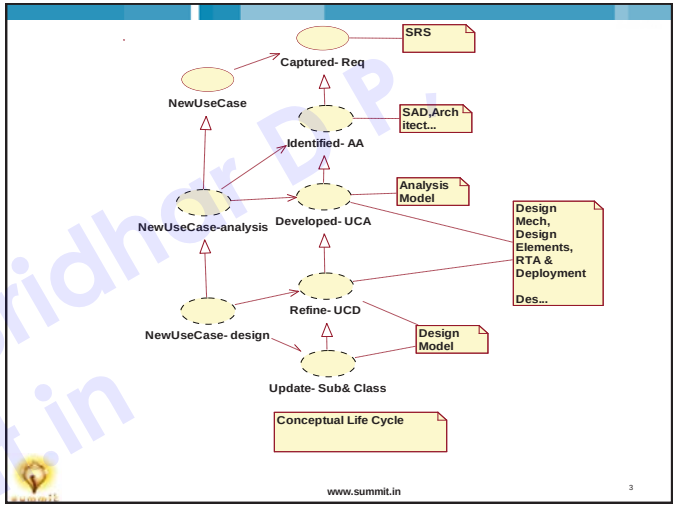
- Lab Exercise
- Check points

www.summit.in 37

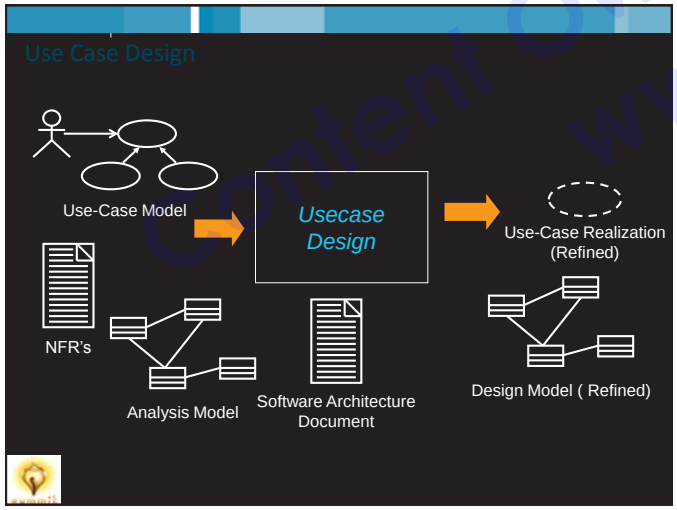
Module 5 : UML in Design Phase

Consulting . Innovation. Mentoring

www.summit.in



www.summit.in

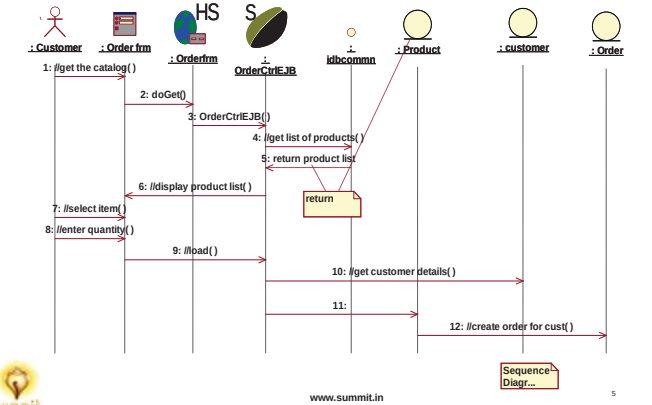


Activities

- To refine use-case realizations in terms of interactions
- To refine requirements on the operations of design classes
- To refine requirements on the operations of design subsystems and/or their interfaces

www.summit.in

Update diagrams



Boundary to Interface Class

- Look for Interface Classes
  - Check if some boundary classes have been converted to Interface classes.
    - Update Analysis Class , Sequence diagrams.
    - Update package , Subsystem & Layers.

Update Patterns

Analysis Mech	Classes	Pattern
Persistency	Order	OODBMS Pattern
Persistency	Invoice	JDBC Pattern
	Product	Observer Pattern

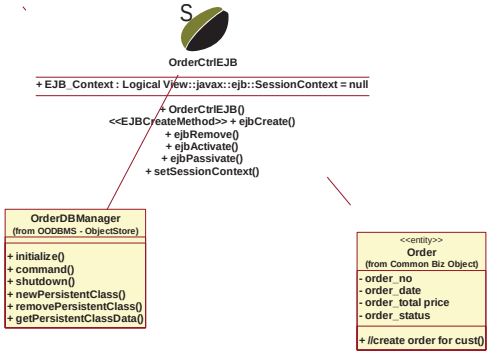
Some more ....

Error Handling

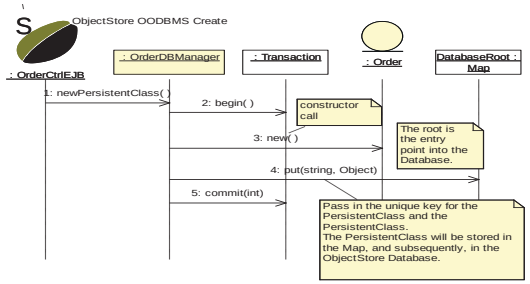
Logging

Messaging

Class Diagram



Apply Patterns



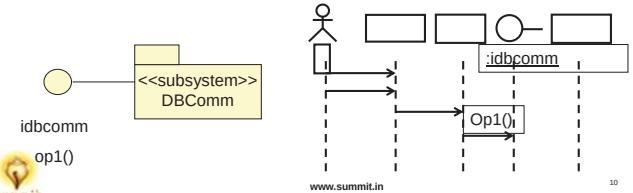
To create a new instance of PersistentClass in the database, the SampleDBManager first creates a transaction and then calls the constructor for PersistentClass. Once the class has been constructed the class is added to the database via the root "put()" operation. The transaction is then committed.

Simplify diagrams

Encapsulate Subflow

- occurs in multiple use-case realizations
- has reuse potential
- is complex and easily encapsulated
- is responsibility of one person/team
- produces a well-defined result

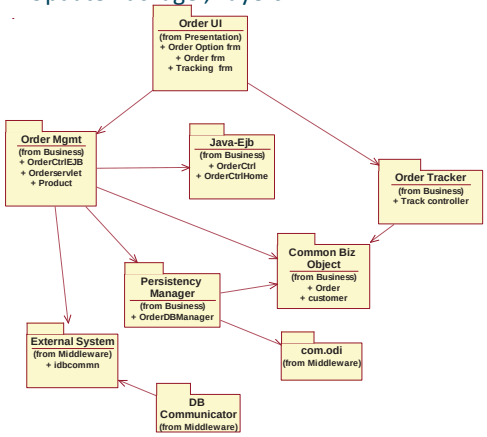
▶ Encapsulate Subsystem Interaction



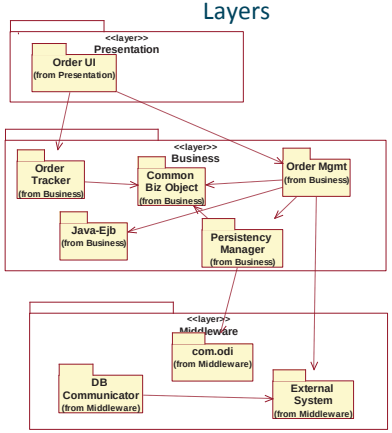
Advantage

- Diagrams are less cluttered
- Can be created before the internal designs of subsystems are created (parallel development)
- Are more generic and easier to change (subsystems can be substituted)

Update Package , Layers



Layers



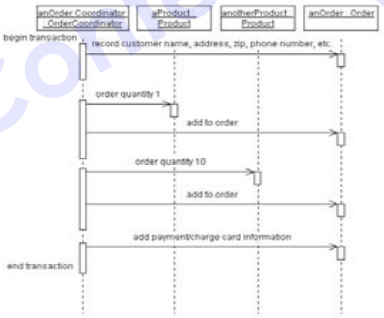
Detailed Flow of Events Description Options

- Comment the interaction diagrams

Scripts can be used to describe the details surrounding these messages.

Script

Note

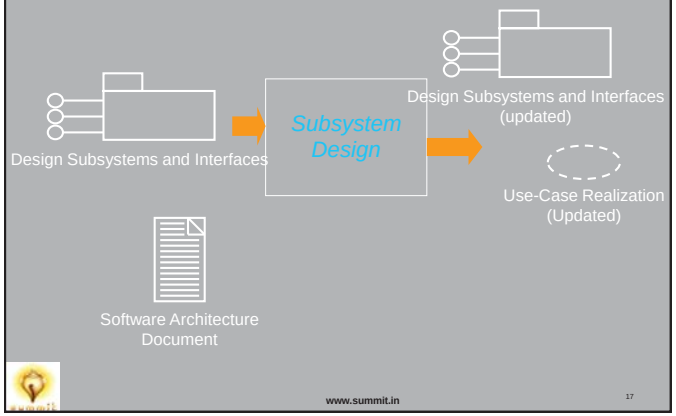


Check Points

- Lab Exercise
- Check points

Module 11 :Subsystem Design

Subsystem Design



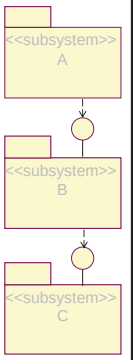
Activities

- To document the internal structure of the subsystem.
- To determine the dependencies upon other subsystems

Subsystem Best Practice

- Goals
  - Loose coupling
  - Portability, plug-and-play compatibility
  - Insulation from change
  - Independent evolution
- Strong Suggestions
  - Don't expose details, only interfaces
  - Only depend on other interfaces

Key is abstraction and encapsulation



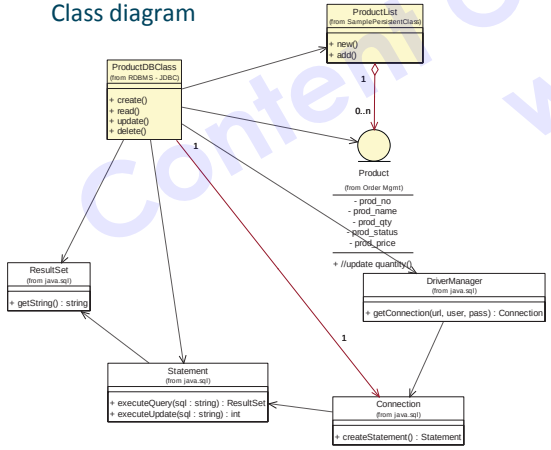
Internal Structure of Subsystem

- The external behavior of a subsystem is primarily defined by the interfaces it realizes. When a subsystem realizes an interface, it makes a commitment to support each and every operation defined by the interface. The operation may be in turn realized by an operation on a design element (i.e., Design Class or Design Subsystem) contained by the subsystem; this operation may require collaboration with other design elements
- Subsystem Usage
  - can be independently ordered, configured, or delivered
  - can be independently developed, as long as the interfaces remain unchanged
  - can be independently deployed across a set of distributed computational nodes
  - can be independently changed without breaking other parts of the systems

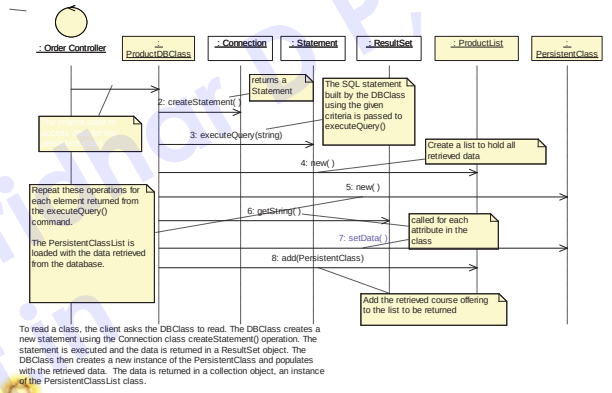
Subsystem Identification Steps

- Make / Buy Decision
  - If Buy, Update your model with third party interfaces.
  - If not, Check for Pattern
    - If a ready made solution is available, update subsystem
- Understand the goal of Subsystem
  - Check interface operation.
  - Identify Classes (same techniques as used in the other modules)
- For Every Interface operation, create interaction diagram
  - Update operations.
- Check to see if there are other subsystems

Class diagram

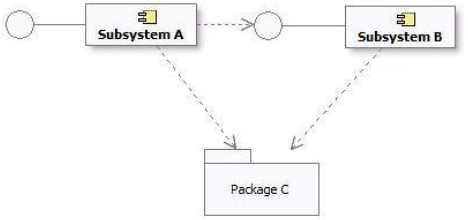


Sequence diagram

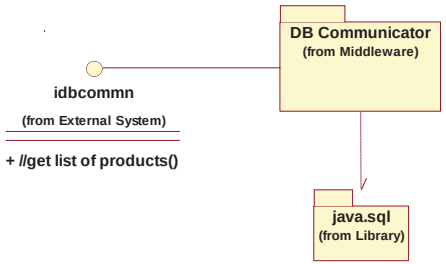


To read a class, the client asks the DBClass to read. The DBClass creates a new statement using the Connection class createStatement() operation. The statement is executed and the data is returned in a ResultSet object. The DBClass then creates a new instance of the PersistentClass and populates with the retrieved data. The data is returned in a collection object, an instance of the PersistentClassList class.

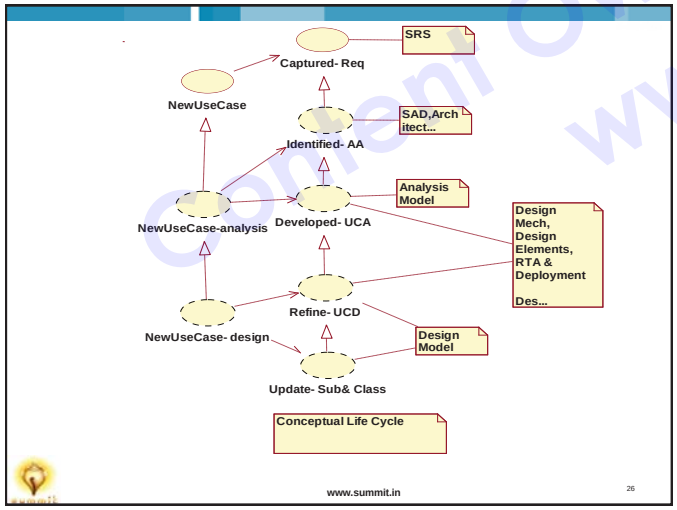
Subsystem - Dependency Package



Dependency



www.summit.in 25

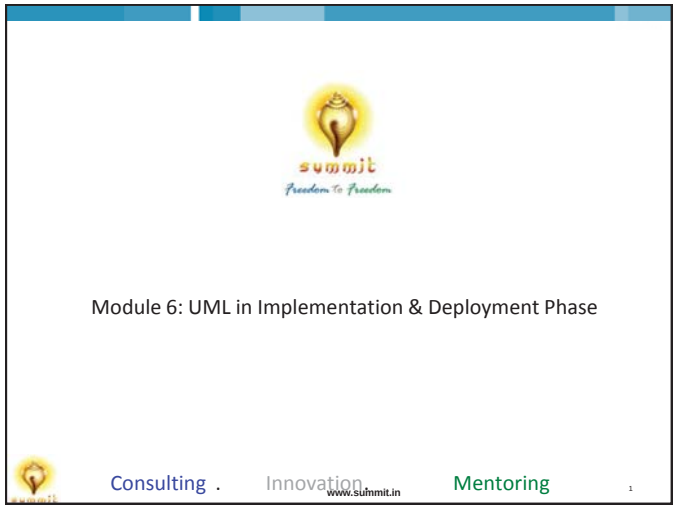


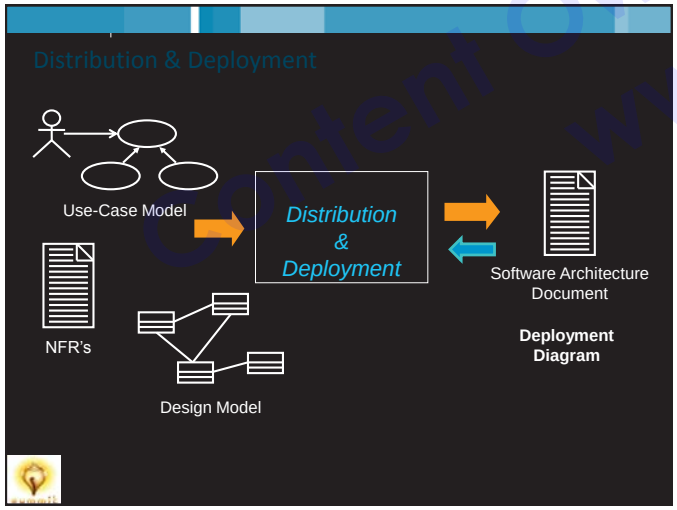
www.summit.in 26

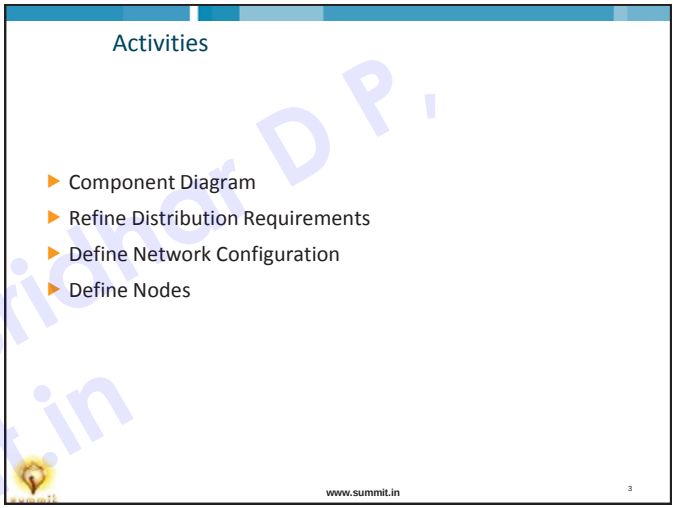
Check Points

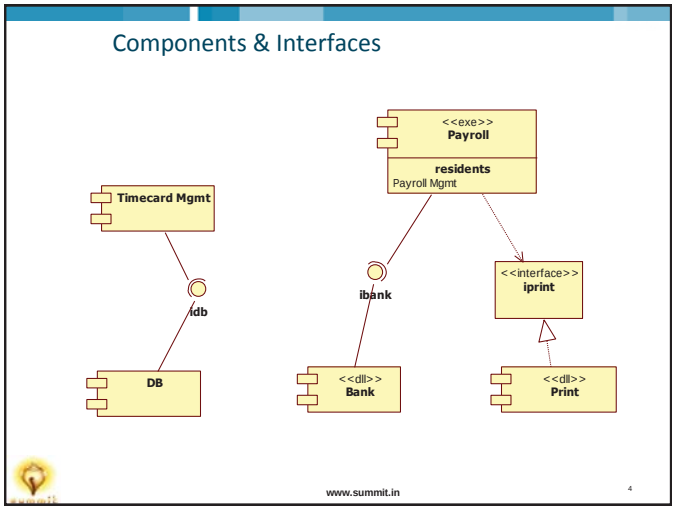
- Lab Exercise
- Check points

www.summit.in 27

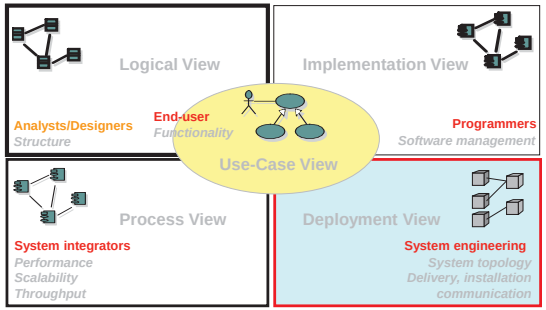








Key Concepts: The Deployment View



Distribution Requirements

- **Distribution demands in the problem domain (functional requirements) –**
  - There may be explicit requirements that the system access or use a specific distributed processor, database, or legacy system to perform part of its functionality.
- **Selected deployment configuration –**
  - Specific deployment configurations impose constraints on the system's distribution by defining the number and types of nodes and their interconnections. For example, selection of a multi-tier deployment configuration typically means that you have a client node, a web server node, and an application server node
  - **Required resources (nonfunctional requirements)** - Time-intensive or computation-intensive functionality might require specific hardware configurations specifically equipped to handle the demands of the functionality; for example, a fast processor, a lot of RAM, or a large amount of disk space.

Distribution Requirements

- **The need for fault tolerance (nonfunctional requirements)** - The requirement could be to have backup processors.
- **Scalability and flexibility concerns (nonfunctional requirements)** - The large numbers of concurrent users are simply too many to support on any single processor. There could be a requirement to load balance the system functionality, thereby providing maximum performance and scalability.
- **Economic concerns** - The price performance of smaller, cheaper processors cannot be matched in larger models.

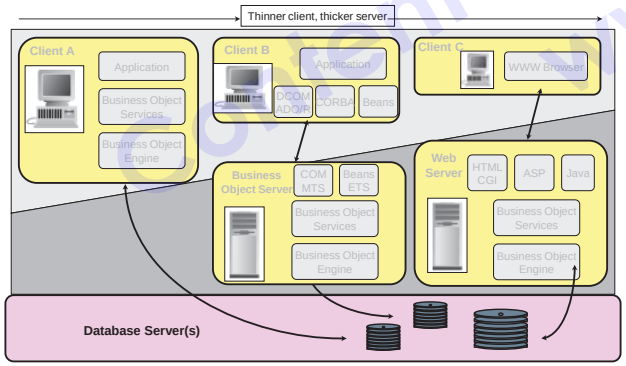
Network Configuration

- The topology of the network.
  - Physical layout of the network, including locations.
  - the nodes on the network, and their configurations and capabilities (the configuration includes both the hardware and the software installed on the nodes, the number of processors, the amount of disk space, the amount of memory, the amount of swap, and so forth) - hardware installed on the node can be represented using devices.
  - the bandwidth of each segment on the network.
  - the existence of any redundant pathways on the network

Architectures

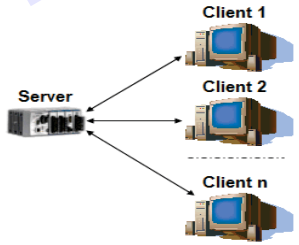
- Two-tier
- Three-tier
- N-tier

Thick to Thin Client



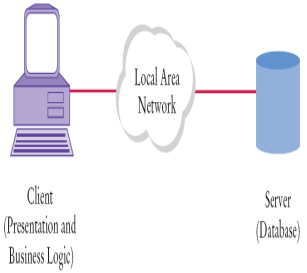
Two-Tier Architecture

- Client connects directly to server
- e.g. HTTP, email



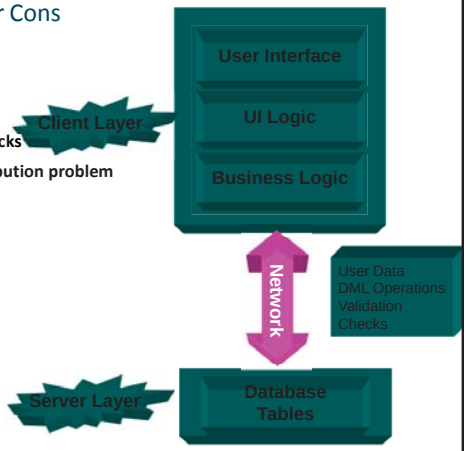
Two-Tier Pros

- Simple
- Client-side scripting offloads work onto the client



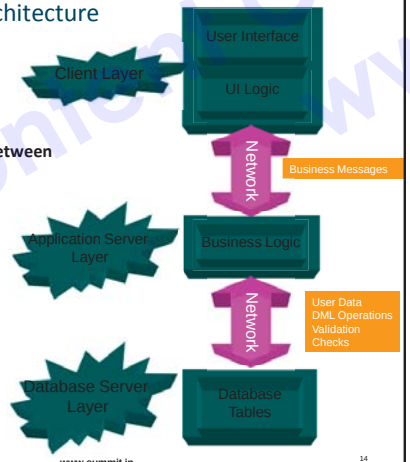
Two-Tier Cons

- Fat client
- Server bottlenecks
- Software Distribution problem
- Inflexible



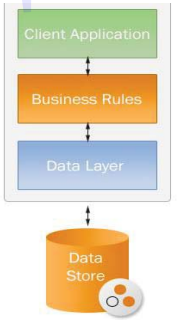
Three-Tier Architecture

- Application Server sits between client and database



Three-Tier Pros

- flexible: can change one part without affecting others
- can connect to different databases without changing code
- specialization: presentation / business logic / data management
- can cache queries

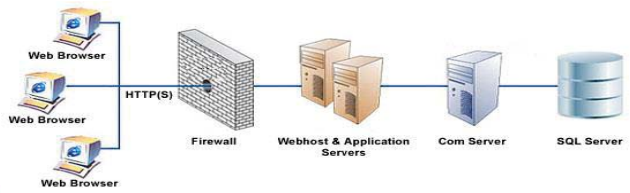


Three-Tier Cons

- higher complexity
- higher maintenance
- lower network efficiency
- more parts to configure (and buy)

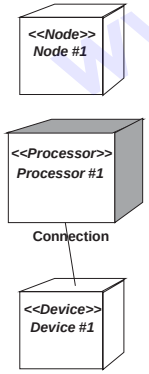
N-Tier Architecture

- Design your application using as many "tiers" as you need
- Use Object-Oriented Design techniques
- Put the various components on whatever host makes sense
- Java allows N-Tier Architecture, especially with RMI and JDBC

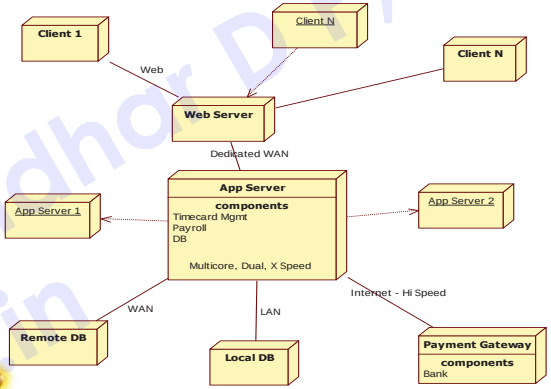


Deployment Model Modeling Elements

- Node
  - Physical run-time computational resource
  - Processor Node
    - Executes system software
  - Device Node
    - Support device
    - Typically controlled by a Processor
- Connection
  - Communication mechanism
  - Physical medium
  - Software protocol



Deployment Diagram



Check Points

- Lab Exercise
- Check points

KnowledgeWorks
REFINING THOUGHTS

Your Learning Partner

ServiceOfferings

Assesments FinishingSchool
CorporateTraining SoftwareDevelopment
e-Learning OnlineTraining
Recruitment

Certifications

Agile PMP TOGAF
6Sigma CBAP ISO27001 Scrum
IIBA ITIL PMI-ACP

Technologies

iOS LTE
Android OOAD RFID Talend VOIP
Hadoop DSP OSS/BSS Systems TOGAF
BigData 3G IPSecurity Telecom/Wireless
Estimation 6Sigma Embedded Python Virtualization
Azure Cloud Perl Ruby Selenium VLSI
ITIL NFC SAP

Clients

Bosch HP Mindtree Philips
Capgemini Mphasis TataElxsi
Aricent Exilant OnMobile Unisys
EMC CISCO Sony
ERICSSON

www.knowledgeworksindia.com