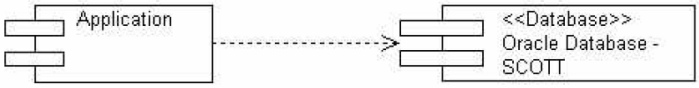
Mapping Object Model to Database Schema:

The data model describes the physical implementation of the database. This is the model of the internal database structure. Persistent classes from the application model map to the data model. The physical data model must map to the database. This is a simple one to one mapping, which is used to forward or reverse engineer the database structures. The mapping between the application model and the database model is more complex. As the data model can change because of normalization or de-normalization, the application model can change as well. Therefore, this mapping must be able to describe any and every possible relationship between the application model and the data model.

Component to Database

The database itself is the physical layer of data storage. It has no mapping into the application model.
Instead, the application has interfaces to the database.
The database is associated to the application. The type of the association is a dependency



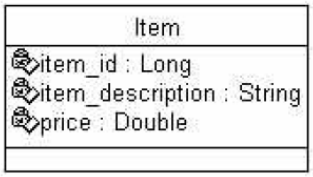
Package to Schema

The persistent view of the application is most commonly modelled in a persistent layer, represented by a package. This package maps to a Schema. The mapping is used for forward and reverse engineering. The mapping, which can be used on a class diagram, is a dependency.

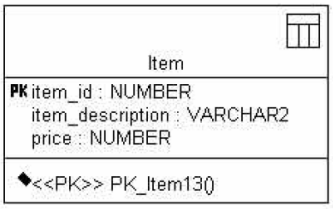
The dependency between package and schema is part of a software design and must be modelled manually.

Classes to Tables

Persistent classes can be mapped to tables. The default mapping is a 1:1 mapping although classes using associations will be in some cases mapped to more than one table. See the "Association to Relationships" section below for details on the mapping of classes associated to another classes. When a class is mapped to a table all of the necessary transformations will be done.
The Item Class, below, is an example of a persistent table, which will be mapped to a database.



The corresponding table is the Item table.

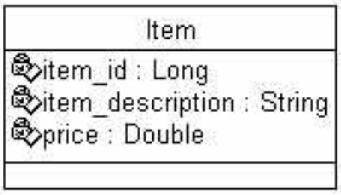


The mapping assigns the table to the class although, as will be seen later, the mapping is additionally provided on the detailed level of a column. The mapping does not require the special design of a class or a table. The forward and reverse engineering itself provides the ability to generate primary keys and primary key constraints.

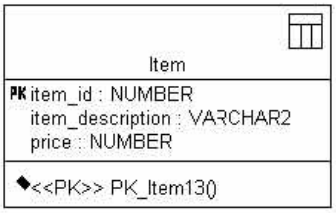
Attributes to Columns

Attributes of persistent classes map to columns of a table. Mapping of attributes must consider the datatype conversion between application datatype and database datatype. Although SQL-92 defines standard datatypes for the database, most vendors implement additional datatypes or change the name of standard datatypes.

The Item class also provides an example of the attribute to column conversion.



The data types used are Long for the id, String for the description and double for the price. In the case of an Oracle 8.x database this mapping is done as follows.



The Long is mapped to NUMBER(10)1, the String to VARCHAR2, and the Double to NUMBER(20).

Associations to Relationships

In a persistent data class model any of the provided association types can be used. To make it even more complex, all of the possible cardinalities of the class roles in the association must be supported.

It is difficult to use relationships in a data model, because the relational data model understands only identifying and non-identifying relationships and the1:N cardinality. The 1:1 cardinality must be forced through constraints.

Unified Approach to OOAD

Unified Modeling Language (UML)

During analysis modeling the user model and structural model views are represented

- Provide insight into the usage scenarios for the system (providing guidance for behavioral modeling
- Establish a foundation for the implementation and environment model views by identifying and describing the static structural elements of the system.

UML is organized into two major design activities:

System design:

Primary objective of UML *system design* is to represent the software architecture

- **conceptual architecture** is concerned with the structure of the static class model and the connections between components of the model

- **module architecture** describes the way the system is divided into subsystems or modules and how they communicate by exporting and importing data
- **code architecture** defines how the program code is organized into files and directories and grouped into libraries.
- **execution architecture** focuses on the dynamic aspects of the system and the communication between components as tasks and operations execute.

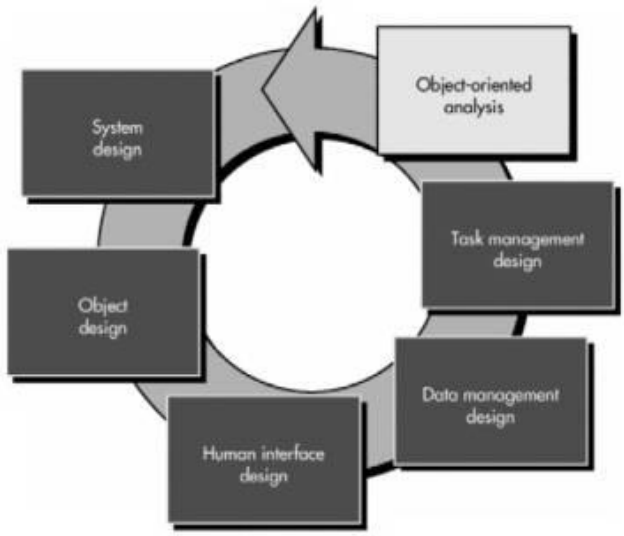
Object design:

- UML *object design* focuses on a description of objects and their interactions with one another.
- detailed specification of attribute data structures and a procedural design of all operations are created.
- visibility for all class attributes is defined and interfaces between objects are elaborated to define the details of a complete messaging model

System and object design are extended to consider:

- **Design of user interfaces**
  - user model view drives the user interface design process
  - provides a scenario that is elaborated iteratively to become a set of interface classes
- **Data management with the system to be built**
  - establishes a set of classes and collaborations that allow the system (product) to manage persistent data (e.g., files and databases)
- **Task management for the subsystems that have been specified.**
  - establishes the infrastructure that organizes subsystems into tasks and then manages task concurrency.

Process flow for OOD



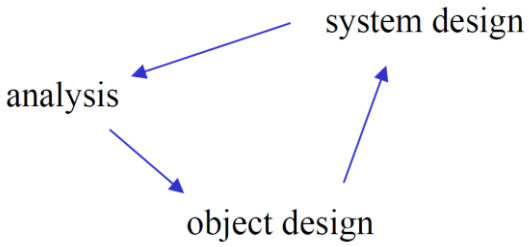
Design methods: Object-Oriented Design
Basic Picture

Basic Picture

Analysis Model	⟷	Design Model
classes	⟷	objects
attributes	⟷	data structures
methods	⟷	algorithms
relationships	⟷	messaging
behavior	⟷	control

Generic components of OO design model
Problem Domain
Subsystems responsible for specific customer requirements.
Human Interaction
Subsystems that implement user interface.
Task Management
Subsystems responsible for controlling and coordinating concurrent tasks.
Data Management
Subsystem responsible for storage and retrieval of objects.

The System Design Process



- System design develops the architectural detail required to build a system or product.
- System design encompasses the following activities:
  1. Partition the analysis model into subsystems.
  2. Identify concurrency that is dictated by the problem.
  3. Allocate subsystems to processors and tasks.
  4. Develop a design for the user interface.
  5. Choose a basic strategy for implementing data management.
  6. Identify global resources and the control mechanisms required to access them.
  7. Design an appropriate control mechanism for the system, including task management.
  8. Consider how boundary conditions should be handled.
  9. Review and consider trade-offs.

1. Partition the Analysis Model

Partition the analysis model to define cohesive collections of classes, relationships and behavior -> **subsystems**

- These design elements are packaged as a **subsystem**.
  - all of the elements of a subsystem share some property in common
  - all may be involved in accomplishing the same function
  - they may reside within the same product hardware, or they may manage the same class of resources

Subsystems are characterized by their responsibilities -> identified by the services that they provide

2. Concurrency and Subsystem Allocation

Dynamic aspects of object-behavior model provide the indication of concurrency among objects or subsystems.
If objects or subsystems must act on events asynchronously and at the same time, they are viewed as concurrent.
When concurrent, there are two options:
1. Allocate to independent processors; or
2. Allocate to the same processor and provide concurrency support through OS features.

3. The Task Management Component

A basic strategy.
Characteristics of tasks are determined by understanding how the task is initiated.
A coordinator task and associated objects are defined.
The coordinator and other tasks are integrated.

A basic task template.
Task name: the name of the object.
Description: purpose of object in narrative form.
Priority.
Services: a list of operations that are object responsibilities.
Coordinates by: the manner in which object behavior is invoked.
Communicates via: input and output data values for the task.

4. The User Interface Component

The user interface represents a critically important subsystem for most modern applications.
OO analysis model contains:

- usage scenarios (called *use-cases*)
- descriptions of the roles that users play (called *actors*) as they interact with the system.
- These serve as input to the user interface design process.

5. The Data Management Component

There are two distinct areas of concern:
1. Management of data critical to the specific application; and
2. Creation of an infrastructure for storage and retrieval of objects.
Often, a DBMS is an appropriate choice for 2. (Relational technology is pervasive.) A DBMS can be used to resolve the following problems and issues:

- Reliability (with backup and recovery);
- Concurrency control; and
- Data independence.

For many tasks/requirements, SQL is much more convenient than other languages.

6. The Resource Management Component

- Different resources are available to an OO system or product
- Subsystems compete for these resources at the same time.
- Global system resources can be external entities (e.g., a disk drive, processor, or communication line) or abstractions (e.g., a database, an object).
- Regardless of the nature of the resource - design a control mechanism for it.

The Object-Oriented Analysis

Analysis is the investigation of the problem - what are we trying to do?
Design is a conceptual solution that meets the requirements – how can we solve the problem?
Object-oriented analysis: Investigate the problem, identify and describe the objects (or concepts) in the problem domain.
Object-oriented design: Considering the results of the analysis, define the software classes and how they relate to each other.
Not every object in the problem domain corresponds to a class in the design model and viceversa.
Where do we assign responsibilities to the objects? Probably a little in both parts.

Iterative Development

Iterative and incremental development is a process that combines the iterative design method with the incremental build model. It is used by software developers to help manage projects.

To fully understand the incremental and iterative development process, you must first split it into its two parts:

Incremental: An incremental approach breaks the software development process down into small, manageable portions known as increments. Each increment builds on the previous version so that improvements are made step by step.

Iterative: An iterative model means software development activities are systematically repeated in cycles known as iterations. A new version of the software is produced after each iteration until the optimal product is achieved.

Iterative and incremental development models are complementary in nature, which is why they are often used together to boost their efficacy and achieve project deliverables.

The Unified Process divides the project into four phases:

1. Inception
2. Elaboration (milestone)
3. Construction (release)
4. Transition (final production release)

Inception Phase

Inception is the smallest phase in the project, and ideally, it should be quite short. If the Inception Phase is long then it may be an indication of excessive up-front specification, which is contrary to the spirit of the Unified Process.

The following are typical goals for the Inception phase:

- Establish
- Prepare a preliminary project schedule and cost estimate
- Feasibility
- Buy or develop it

The Lifecycle Objective Milestone marks the end of the Inception phase.
Develop an approximate vision of the system, make the business case, define the scope, and produce rough estimate for cost and schedule.

Elaboration phase

During the Elaboration phase, the project team is expected to capture a healthy majority of the system requirements. However, the primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture. Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams).

The architecture is validated primarily through the implementation of an Executable Architecture Baseline. This is a partial implementation of the system which includes the core most architecturally significant components. It is built in a series of small time-boxed iterations. By the end of the Elaboration phase, the system architecture must have stabilized and the executable architecture baseline must demonstrate that the architecture will support the key system functionality and exhibit the right behavior in terms of performance, scalability, and cost.

The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the Construction phase. At this point the plan should be accurate and credible since it should be based on the Elaboration phase experience and since significant risk factors should have been addressed during the Elaboration phase.

Construction phase

Construction is the largest phase of the project. In this phase, the remainder of the system is built on the foundation laid in Elaboration. System features are implemented in a series of short, time-boxed iterations. Each iteration results in an executable release of the software. It is customary to write full-text use cases during the construction phase and each one becomes the start of a new iteration. Common Unified Modeling Language (UML) diagrams used during this phase include activity diagrams, sequence diagrams, collaboration diagrams, State Transition diagrams and interaction overview diagrams. Iterative implementation for the lower risks and easier elements are done. The final Construction phase deliverable is software ready to be deployed in the Transition phase.

Transition phase

The final project phase is Transition. In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations. The Transition phase also includes system conversions and user training.

Object-Oriented Testing

Testing is a continuous activity during software development. In object-oriented systems, testing encompasses three levels, namely, unit testing, subsystem testing and system testing.

Unit Testing

In unit testing also known as class resting, the individual classes are tested. It is seen whether the class attributes are implemented as per design and whether the methods and the interfaces are error-free. Unit testing is the responsibility of the application engineer who implements the structure.

Subsystem Testing

Subsystem Testing also known as integration or inter-class testing involves testing a particular module or a subsystem and is the responsibility of the subsystem lead. It involves testing the associations within the subsystem as well as the interaction of the subsystem with the outside. Subsystem tests can be used as regression tests for each newly released version of the subsystem.

System Testing

System testing involves testing the system as a whole and is the responsibility of the quality-assurance team. The team often uses system tests as regression tests when assembling new releases.

Object-Oriented Testing Techniques

The different types of test cases that can be designed for testing object-oriented programs are called **grey box test cases**. Some of the important types of grey box testing are −

- **State model-based testing** − This encompasses state coverage, state transition coverage and state transition path coverage.
- **Use case-based testing** − Each scenario in each use case is tested.
- **Class diagram-based testing** − Each class, derived class, associations and aggregations are tested.
- **Sequence diagram-based testing** − The methods in the messages in the sequence diagrams are tested.

Techniques for Subsystem Testing

The two main approaches of subsystem testing are −

- **Thread based testing** − All classes that are needed to realize a single use case in a subsystem are integrated and tested.
- **Use based testing** − The interfaces and services of the modules at each level of hierarchy are tested. Testing starts from the individual classes to the small modules comprising of classes, gradually to larger modules, and finally all the major subsystems.

Categories of System Testing

- **Alpha testing** − This is carried out by the testing team within the organization that develops software.
- **Beta testing** − This is carried out by select group of co-operating customers.
- **Acceptance testing** − This is carried out by the customer before accepting the deliverables.

Challenges in Testing Object-oriented Programs

Traditional testing methods are not directly applicable to OO programs as they involve OO concepts including encapsulation, inheritance, and polymorphism. These concepts lead to issues, which are yet to be resolved. Some of these issues are listed below.

1. Encapsulation of attributes and methods in class may create obstacles while testing. As methods are invoked through the object of corresponding class, testing cannot be accomplished without object. In addition, the state of object at the time of invocation of method affects its behavior. Hence, testing depends not only on the object but on the state of object also, which is very difficult to acquire.
2. Inheritance and polymorphism also introduce problems that are not found in traditional software. Test cases designed for base class are not applicable to derived class always (especially, when derived class is used in different context). Thus, most testing methods require some kind of adaptation in order to function properly in an OO environment.