Object-Orientation

Wolfgang Emmerich

Outline

- Problem of software system complexity
- Approaching a solution
  - Human limitations
  - Underlying principles
- Development of the object-oriented approach
- Assessment of object-orientation

Software System Complexity

- *"Consisting of or comprehending various parts united or connected together; formed by a combination of different elements"*
- *Any sizeable system:*
  - *developed by a team in a lengthy process,*
  - *impossible for individual to comprehend fully,*
  - *difficult to document and test,*
  - *potentially inconsistent or incomplete,*
  - *subject to change.*
- *But : Software engineering cannot yet provide fundamental laws to explain*

Reasons for Complexity

- *Grady Booch's four reasons for complexity of software-intensive systems:*

1 Nature of the problem domain

- *requirements,*
- *decay of systems*

2 Complexity of process

- *management problems,*
- *need for simplicity*

Reasons for Complexity (continued)

3 Dangerous potential for flexibility of software

- *"Software is flexible and expressive and thus encourages highly demanding requirements, which in turn lead to complex implementations which are difficult to assess"*

4 Characterising behaviour of systems

- *"The task of the software development team is to engineer the illusion of simplicity"*

Systems and Subsystems

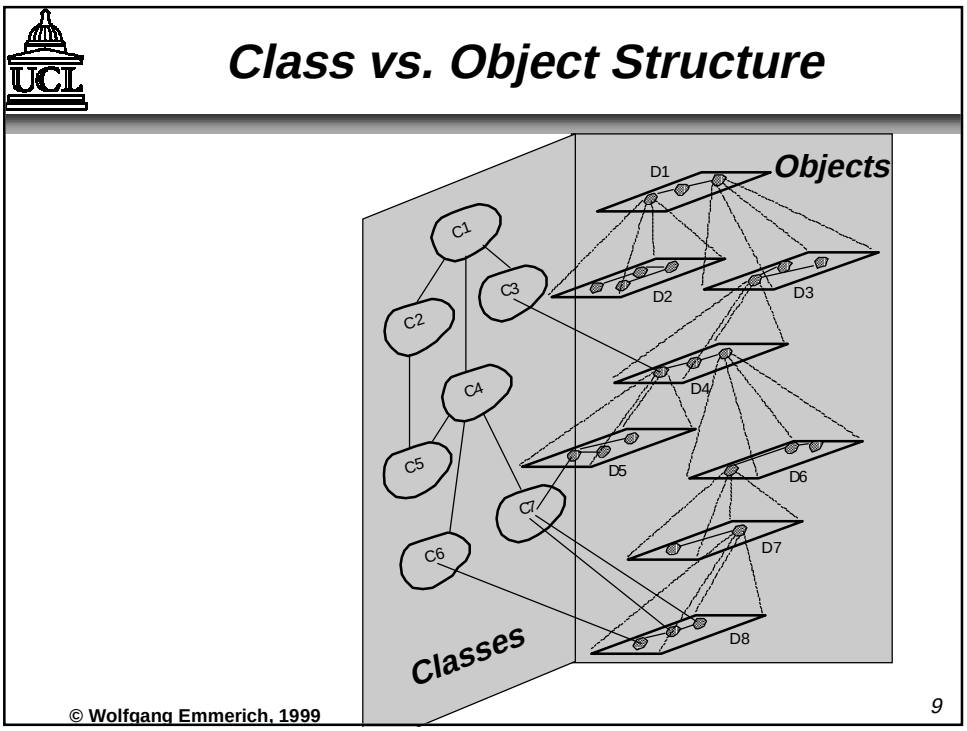
- *"An organised or connected group of objects; a whole composed of parts in orderly arrangement according to some scheme or plan"*
- *System:*
  - *boundary*
  - *environment*
  - *character*
  - *emergent property*
- *Systems: interconnected subsystems*
- *Potential for inconsistency and*

Attributes of Complex Systems

- *Booch's five attributes of a complex system:*
  - *Hierarchical and interacting subsystems*
  - *Arbitrary determination of primitive components*
  - *Stronger intra-component than inter-component links*
  - *Combine and arrange examples of a few kinds of subsystems*
  - *Evolution from simple to complex working systems*

Simplifying Complex Systems

- *Usefulness of abstractions common to similar activities, e.g. driving different kinds of motor vehicle*
- *Multiple orthogonal hierarchies e.g. structure and control system*
- *Prominent hierarchies in object-orientation*
  - *"class structure"*
  - *"object structure"*
- *e. g. engine types, engine in a specific car*



Class vs. Object Structure

Approaching a Solution

- *Hampered by human limitations:*
  - *dealing with complexities*
  - *memory*
  - *communications*
- *Principles that will provide basis for development:*
  - *Abstraction*
  - *Hierarchy*
  - *Decomposition*

Abstraction & Hierarchy

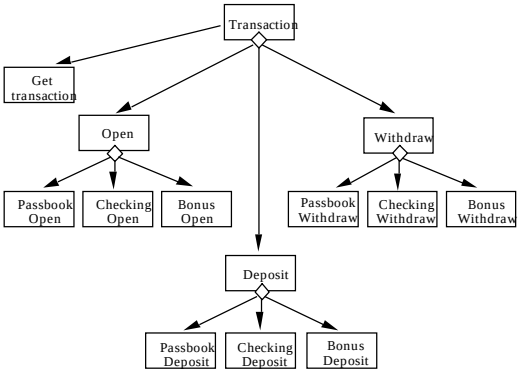
- *Concepts of fundamental importance*
- *Abstraction: assists people's understanding*
  - *grouping,*
  - *generalising,*
  - *'chunking'*

  of information or ideas.
- *Hierarchy:*
  - *Recognition of higher and lower orders,*
  - *Accumulation of attributes at higher level,*
  - *Association of fewer attributes with lower level and greater number*

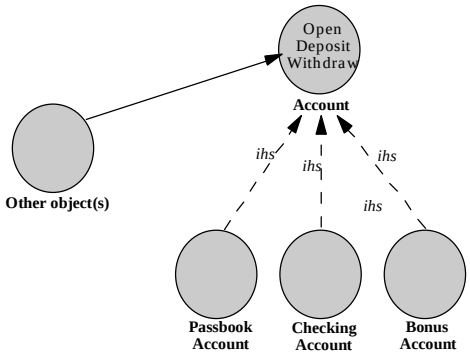
Decomposition

- *Handling complexity on the principle of 'divide and conquer'*
- *Two forms of decomposition:*
  - *process-oriented: according to steps / functions*
  - *object-oriented: according to behaviour of autonomous objects*
- *Both valid, but current claims for superiority of OO*
  - *stronger framework*
  - *reuse of common abstractions*
  - *resilient under change*

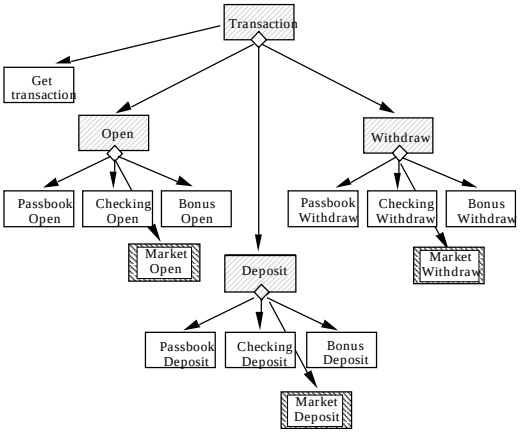
A function/data composition



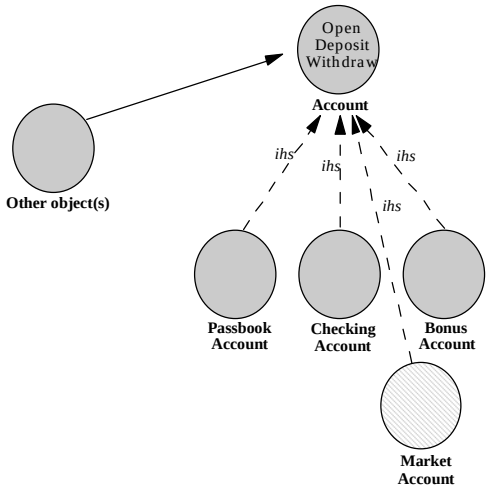
An object-oriented Decomposition



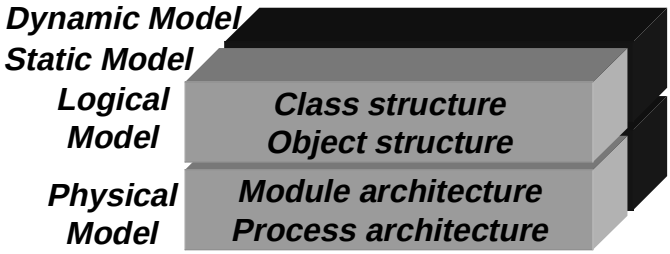
Adding a new Market Account



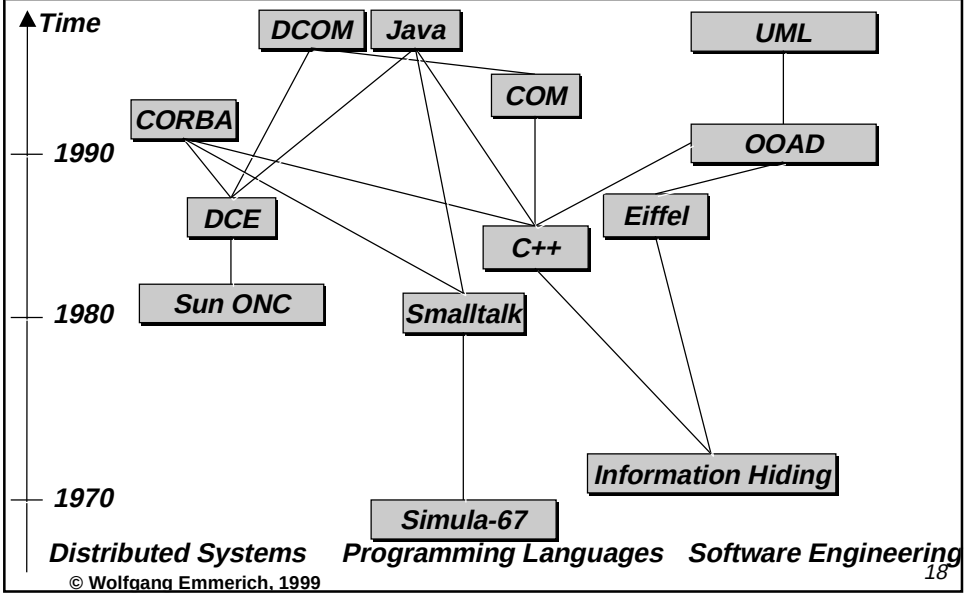
Adding a new Market Account



Model of OO Development



History of Object-Orientation



Structured Methods

- *SSADM (Cutts 1987), SA (de Marco 1978), SADT (Ross 1977).*
- *Existing structure methods treat separately :*
  - *functions (behaviour) and*
  - *data (information held)*
- *Problems:*
  - *Difficulties with maintenance (because need knowledge of data storage)*
  - *Division of knowledge (whereby "what" is transformed into "how")*
- *Instability of functions*

Object-Oriented Methods

- *Better able to cope with change*

Item	Freq. Of Changes
Object from application	Low
Long-lived information structures	Low
Passive object's attribute	Medium
Sequence of behavior	Medium
Interface with outside world	High
Functionality	High

- *OO focuses analysis on problem domain*
- *Promotes reuse*
- *Continuity of representation*

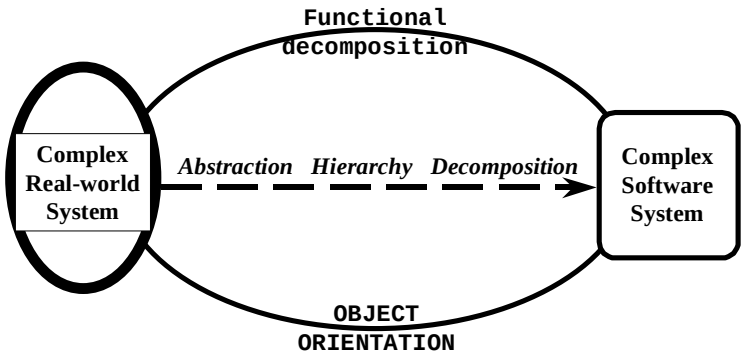
Suggested object-oriented Methods

- Coad & Yourdon (91) for OOA
- Booch (94) also for OOA
- Jackson (83) for system design
- Jacobson (92) OOSE
- Approach of this course based on Jacobson because employs 'use cases' throughout
  - essential user role
  - focus on domain
  - integration in process
- All likely to be superseded by 'retreads' *derivatives of the UML*

Drawbacks of OO

- Large scale reuse not yet achieved
- Few available reusable libraries
- Managing reusable libraries is a problem
- Extensive retraining before pervasive

Summary



Object-Orientation (Cont'd)

Wolfgang Emmerich

Lecture Overview

- *Basic principles of object-orientation*
  - *objects*
  - *classes of object*
  - *encapsulation*
  - *inheritance*
  - *polymorphism*
- *Object-orientation vs. structured methods*

What's object-orientation all about ?

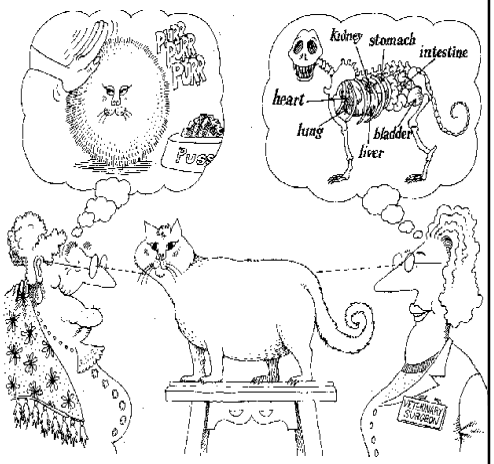
- *Principles and techniques for system modelling which:*
  - *aim to produce a model of a system*
  - *provide notations and methods*
- *Advantages for software development:*
  - *reduces the 'semantic gap' between reality and models*
  - *makes system understanding easier*
  - *allows local modification to models*

An Object

- *An object is: "An abstraction of something in a problem domain, reflecting the capabilities of a system to keep information about it, interact with it, or both" (Coad & Yourdon 91)*
- *"An entity able to save a state (information) and which offers a number of operations (behaviour) to either examine or affect this state" (Jacobson 92)*

Objects & Abstraction

- *"An abstraction denotes the essential characteristics of*
- *an object that distinguish it from all other kinds of objects*
- *and thus provide crisply defied conceptual boundaries,*
- *relative to perspective of viewer*



Sample Objects

- From a “Food Manufacturing Company”
- Passive objects :
  - one individual sack of lentils
  - invoice 63501 sent to A Farm, Lincolnshire
- Active objects :
  - lorry "M235 BCM"
  - van "N683 CNM"
- Human agents :
  - Richard Green
  - David Brown (Executive)
  - Hill, D (Truck driver)
- Structure objects :
  - Marketing Department

Attributes and Associations

- Any object has both attributes and associations
- Attributes characteristic features or properties name / value pairs
- Association any kind of link or connection between one object and one or a set of other objects
- Characteristics private to an object best represented as attributes

Associations are Relationships

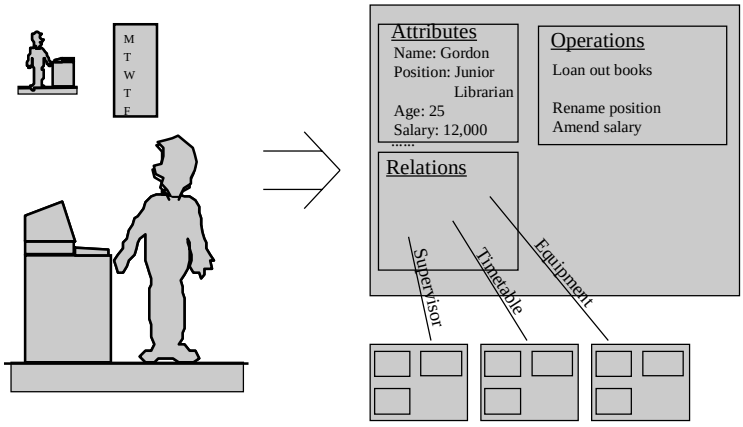
- *Two essential oppositions :*
- *aggregation vs reference relationships*
  - *aggregation relationships where connection creates composite objects from simple objects*
  - *reference relationships where connection only refers to another object*
- *static vs dynamic relationships*
  - *static relationships, where coupling of objects is stored over a long period of time,*
  - *dynamic relationships, which are established by operations*

Viewpoint of Association

- *View and purpose affect definition of assoc.*
- *Partition or division: Association indicating new objects created by splitting other objects apart , e.g. A book 'consists of' or 'can be divided into': title page, introduction, chapters, conclusion, index*
- *Aggregation or amalgamation: Association indicating new objects created by adding other objects together e.g Covers, binding, and end papers are*

Structure and Behaviour of an Object

- *Gordon, Junior Librarian, as object*



Behaviour and Structure

- *Example in a Food Company*

One packet of herbal tea

Attributes
Price: 2.95
Weight: 1kg

Relations
Order: 57891
Delivery:
Supplier: A Farm

Operations
Is Stored
Is Sold
Is Delivered

Richard Green

Attributes
Age: 45
Sex: Male

Relations
Managing Director

Operations
Decide policies
Attend meetings
Manage Board meetings

Lorry M235 BCN

Attributes
Reg: M235 BCN
Weight: 7.5 tons
Fuel level: 40 litres

Relations
Driver Hill, D
Trailor SSFG12

Operations
Deliveries Goods
Repeat Until End
Go To Destination
Unload Goods
EndRepeat
Refuel....

Marketing Department

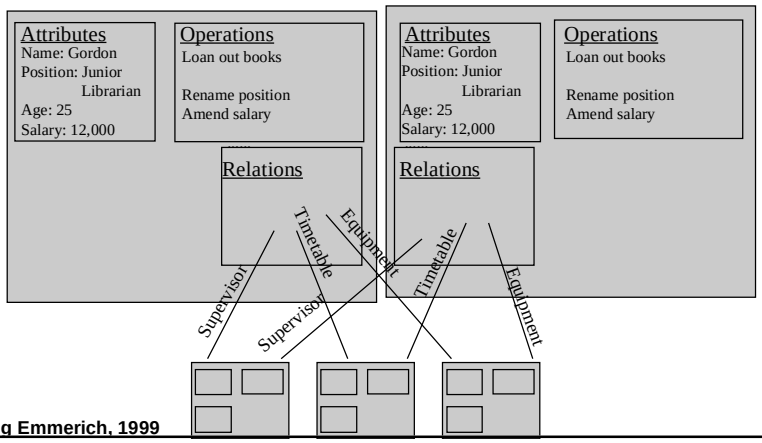
Attributes
Staff: 50 people
Location: High St

Relations
Marketing Director
Marketing Team

Operations
Market All Products
Choose New Products
Determine Market Niches

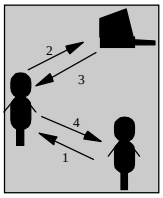
Object Identity

- **Separate objects each have a unique object identity**



Dynamics of Objects

- **Dynamics are generated through stimuli or messages passing between objects**
  - **Receipt of a stimulus cause operation by (or in) the receiving object**
  - **Receipt of a stimulus can trigger sending of another stimulus to other objects**



Encapsulation

- *"Behaviour and information are encapsulated in objects" (Jacobson 1992)*
- *"Encapsulation is the process of compartmentalising the elements of an abstraction that constitute its structure and behaviour" (Booch 1995)*
  - *Only the 'interface' of an object is 'visible' to other objects*
  - *Need, and can, only know operations on an object, not how they work nor about other characteristics*
  - *Necessary prerequisite for information hiding*

Encapsulation (cont'd)

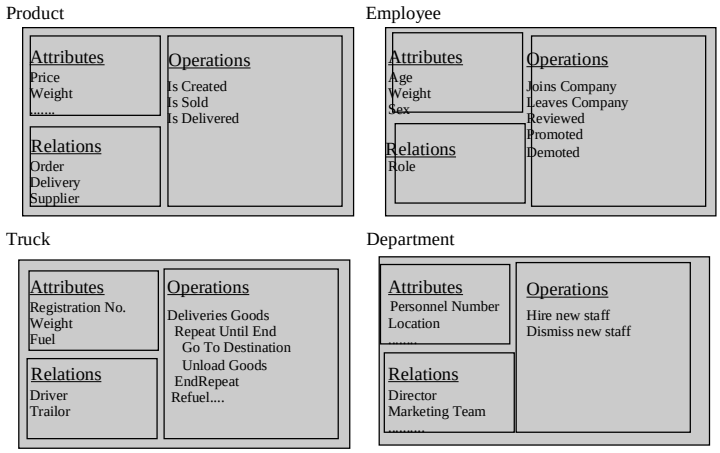
- *Encapsulation and abstraction are complementary concepts*
  - *Abstraction focuses on the observable characteristics and behaviour of the object*
  - *Encapsulation focuses on the representation derived from these characteristics*
- *Encapsulation requires :*
  - *explicit division between abstractions*
  - *clear separation of their concerns*

Classes of Objects

- *Classes represent groups of objects which have the same behaviour and information structures.*
- *Class is a kind of type, an ADT (but with data), or an 'entity' (but with methods)*
- *Classes are the same in analysis and design*
- *"A class represents a template for several objects ...Objects of the same class have the same definition both for their operations and for their information*

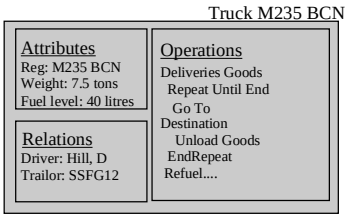
Sample Classes

- *From a Food Company*



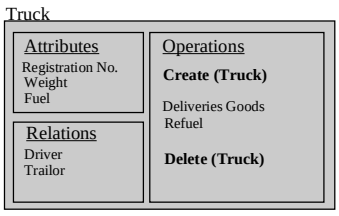
Objects are Instances of Classes

- *Every object is an instance of a single class*
- *A class defines the possible behaviours and the information structure of all its object instances.*
- *Different instances may have their operations activated in different ways and in different sequences; hence they may be in different states.*



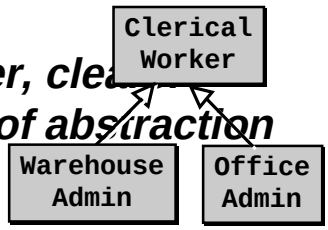
Instantiation of objects

- *Instantiation of a class generates an object, an instance of its class*
- *Instantiation demands a specific create operation in every class*



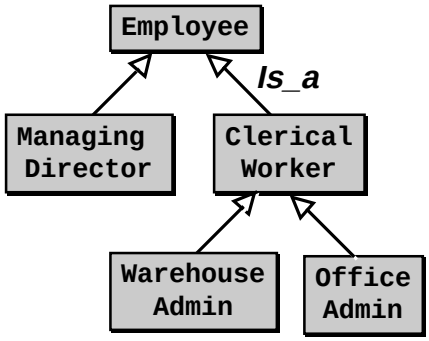
Inheritance

- *Inheritence: a relationship between different classes with common characteristics*
- *"If class B inherits class A, then both the operations and information structure in class A will become part of class B" (Jacobsen 92)*
- *Major benefits are simpler, clea... classes, at higher levels of abstraction*



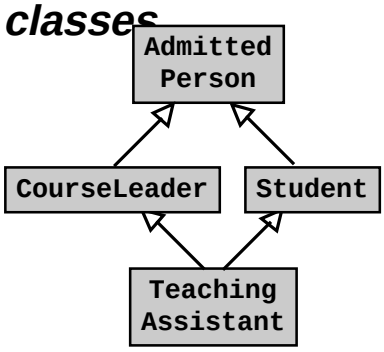
Generalisation & Specialisation

- *Generalisation - Creation of an 'ancestor'*
- *Specialisation - Creation of a 'descendant'*



Multiple Inheritance

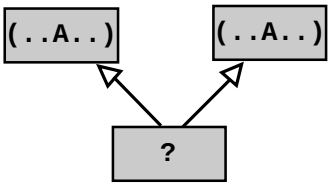
- *One class inherits from two or more existing classes*



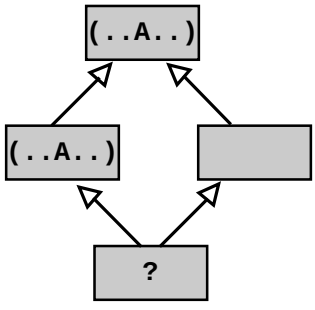
- *Allows more complex class structures, but*
- *less easily understood*

Multiple Inheritance & Ambiguities

- *Name clashes :*



- *Incorrect repeated inheritance*



Polymorphism

- *System behaviour is defined by the dynamic behaviour of instances of classes*
- *"Polymorphism means that the sender of a stimulus does not need to know the receiving instance's class. The receiving instance can belong to an arbitary class" (Jacobsen 1992)*
- *Polymorphism enables different instances of different classes to be associated*
- *A receiving instance interprets stimuli according to its own class*

Structured Methods

- *SSADM (Cutts 1987), SA (de Marco 1978), SADT (Ross 1977).*
- *Existing structured methods treat separately*
  - *functions (behaviour) and*
  - *data (information held)*
- *Problems:*
  - *Difficulties with maintenance (because need knowledge of data storage)*
  - *Division of knowledge (whereby "what" is transformed into "how")*
- *Instability of functions*

Object-Oriented Methods

- **Better able to cope with change**

Item	Freq. Of Changes
Object from application	Low
Long-lived information structures	Low
Passive object's attribute	Medium
Sequence of behavior	Medium
Interface with outside world	High
Functionality	High

- **OO focuses analysis on problem domain**
- **Promotes reuse**
- **Continuity of representation**

Suggested object-oriented Methods

- **Coad & Yourdon (91) for OOA**
- **Booch (94) also for OOA**
- **Jackson (83) for system design**
- **Jacobson (92) OOSE**
- **Approach of this course based on Jacobson because employs 'use cases' throughout**
  - **essential user role**
  - **focus on domain**
  - **integration in process**
- **All likely to be superseeded by 'retreads' based on the UML**

Drawbacks of OO

- *Large scale reuse not yet achieved*
- *Few available reusable libraries*
- *Managing reusable libraries is a problem*
- *Extensive retraining before pervasive*

Summary

- *Basic concepts:*
  - *object: entity combining essential characteristics abstracted from a domain*
  - *class: expression of objects' common characteristics*
  - *encapsulation: combination of attributes & operations in a single self-contained object*
  - *inheritance: relationship between super-class and sub-class defining levels of commonality*
  - *polymorphism: facility allowing stimuli to ignore class of receiving object*

Summary (Cont'd)

- *Application of concepts enables:*
  - *object identification*
  - *object modelling*
  - *behaviour modelling*

D50: Advances in Software Engineering

Requirements Model

Wolfgang Emmerich

Lecture Overview

- *Object-Oriented Software Engineering (OOSE) from Jacobson et al.*
- *The basics of 'a use case driven approach'*
- *Development of a Requirements Model:*
  - *actors*
  - *use cases*
  - *interface descriptions*
  - *problem domain objects*
- *Relevant notations from the UML (Unified Modeling Language)*

OOSE Background

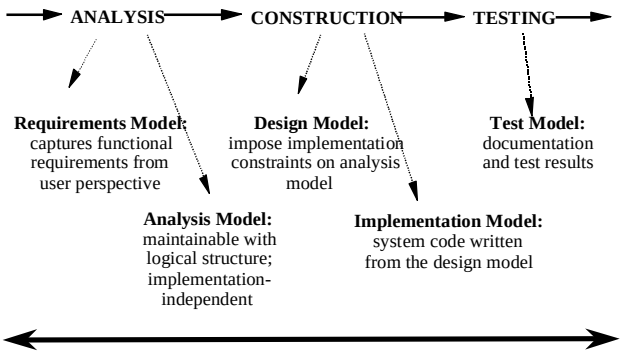
- *Originated in Sweden*
- *"Object-Oriented Software Engineering - A Use Case Driven Approach" by Ivar Jacobson, Magnus Christerson, Patrik Jonsson & Gunnar Overgaard, Addison-Wesley , 1992*
- *Pragmatic method based on experience*
- *Popular and successful*
- *Complete method*

What comprises a Method?

- ***Method described via***
  - ***syntax (how it looks)***
  - ***semantics (what it means)***
  - ***pragmatics (heuristics, rules of thumb for use)***

System Dev. as Building Models

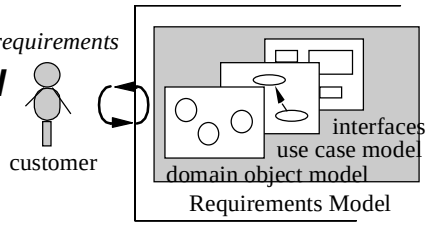
- ***3 stages and 5 models***



Seamless, incremental transition between stages and models, iterations possible

Analysis Stage

- **Primary objectives**
  - to determine what the system must do
  - to embed the software system in its environment
- **Two concerns**
  - to get the right thing
  - to get the thing right (now and for future)
- **Products**
  - Requirements Model
  - Analysis Model



Producing A Requirements Model

- **Inputs**
  1. Derive possible use cases
  2. Discriminate between possible use cases
  3. Generate use case desciptions
  4. Identify associations between use cases
  5. Refine & complete use cases & use case model
  6. Describe and test user interfaces
  7. Describe system interfaces
  8. Identification of problem domain objects
  9. Check incorporation of requirements
- **Outputs**
- **Notations**

Requirements Model: Input & Output

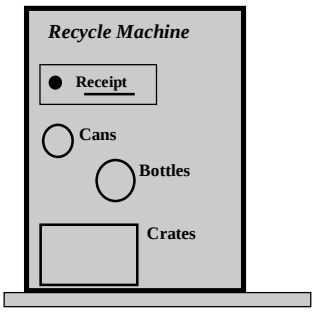
- **Inputs :**
  - User requirements specifications [multi media]
  - Documentation of existing systems, practices etc. that are to be followed [text, graphic]
  - Exchanges between developers and users and specifiers [m m]
- **Outputs :**
  - use case model [graphic]
  - concise descriptions of use cases [text]
  - user interface descriptions [text ... prototypes]
  - system interfaces [protocols]
  - problem domain object list (names, attrib.)

Requirements Model: Notations

- **Notations introduced :**
  - use case diagram (system box, ellipses, names, actor icons, actor/case links, <uses> and <extends> associations)
  - association (<extends>, <uses>)
  - use case descriptions

Requirements Example

Multi-purpose recycling machine



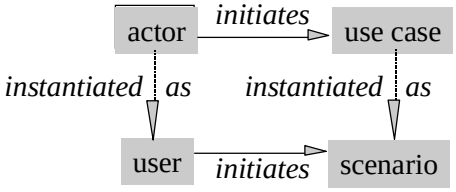
Machine must:
- receive & check items for customers,
- print out receipt for items received,
- print total received items for operator,
- change system information,
- signal alarm when problems arise.

Actors

- ***An actor is:***
  - ***anything external to the system, human or otherwise***
  - ***a user type or category***
- ***A user doing something is an occurrence of such a type***
- ***A single user can instantiate several different actor types***
- ***Actors come in two kinds:***
  - ***primary actors, using system regularly***
  - ***secondary actors, enabling primary actors***

A Use Case

- ***A use case***
  - ***constitutes complete course of events initiated by actor***
  - ***defines interaction between actor and system***
  - ***is a member of the set of all use cases which***
- ***Use cases together define all existing ways of using the system***

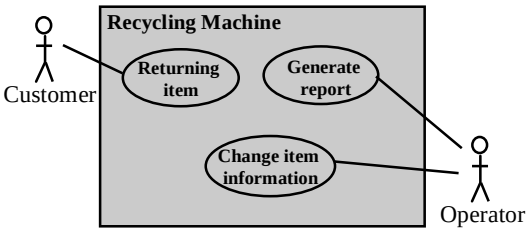


Example Use Case

- ***Returning items is started by Customer when she wants to return cans, bottles or crates. With each item that the Customer places in the recycling machine, the system will increase the received number of items from Customer as well as the daily total of this particular type. When Customer has deposited all her items, she will press a receipt button to get a receipt on which returned items have been printed, as well as the total return sum.***

Use Case Model

- *A use case model*
  - *presents a collection of use cases*
  - *characterise behaviour of whole system, plus external actors*



Identifying Use Cases

- *Consider situation,*
- *Identify actors,*
- *Read specification,*
- *Identify main tasks,*
- *Identify system information,*
- *Identify outside changes,*
- *Check information for actors,*
- *Draft initial use cases [text]*
- *Identify system boundary,*
- *Draft initial use case model [graphic]*

When is a Use Case .... ?

- **Discrimination between possible use cases**
  - **Estimate frequency of use,**
  - **Examine degree of difference between cases**
  - **Distinguish between 'basic' and 'alternative' courses of events**
  - **Create new use cases where necessary**

Elaborated Example

BASIC - When the Customer returns a deposit item, it is measured by the system. The measurements are used to determine what kind of can, bottle or crate has be deposited. If accepted, the Customer total is incremented, as is the daily total for that specific item type.

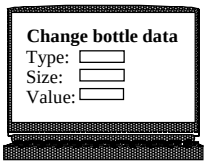
ALTERNATIVE - If the item is not accepted, 'NOT VALID' is highlighted on the panel.

BASIC - When Customer presses the receipt button, the printer prints the date. The customer total is calculated and the following information printed on the receipt for each item type: name, number returned, deposit value, total for this type. Finally the sum that the Customer should receive is printed on the receipt.

User Interface Descriptions

- *Describe user interfaces*
- *Test on potential users,*
- *if necessary using*
- *simulations or prototypes*

Operator's interface



Problem Domain Objects

- *Object in specification have direct counterpart in the application environment*
- *System knowledge obligatory*
- *Refinement in stages :*
  - *Object noun* $\rightarrow$
  - *Logical attributes* $\rightarrow$
  - *Static associations* $\rightarrow$
  - *Inheritance* $\rightarrow$
  - *Dynamic associations* $\rightarrow$
  - *Operations*

Object Examples

Object name type	Attribute characteristic/information :
Deposit item	name: string, total: integer, value: Euro
Can	width: cm, height: cm
Bottle	width: cm, height: cm, bottom: cm
Crate	width: cm, height: cm, lenght: cm
Receipt	total cans: int, total bottles: int, ...
Customer panel	receipt button: button
Operator panel	bottle data: cm, ...

Summary

- System development as model building
- Requirements model “to get the right thing”
- System use in context via the use case model
- User interface descriptions
- Problem domain objects as prelude to class diagram