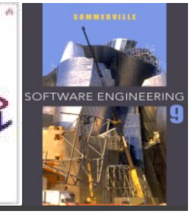
 

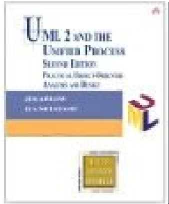
Analysis and Design

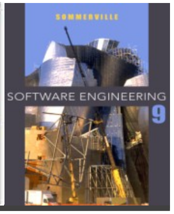
Lecture 4

Outline


- Software analysis and design
- Structured vs. object-oriented methods

- Object-oriented analysis in UML
- Objects and classes
- Finding analysis classes

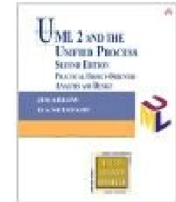


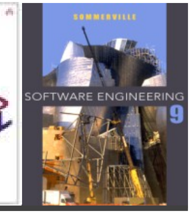


Software Analysis and Design

Lecture 4/Part 1

Analysis, design and implementation



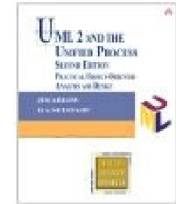


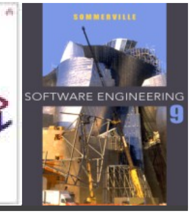
✧ Software development (i.e. analysis, design and implementation) is the stage in the software engineering process at which an executable software system is developed.

✧ Software analysis, design and implementation are invariably inter-leaved with blurred border in between.

- **Software analysis** is a creative activity in which you identify software processes, entities (objects) and their relationships.
- **Software design** refines analytical models with implementation details.
- **Implementation** is the process of realizing the design as a program.

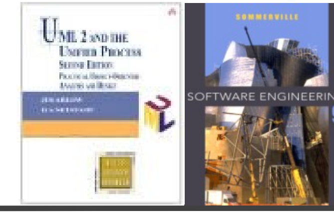
Process stages





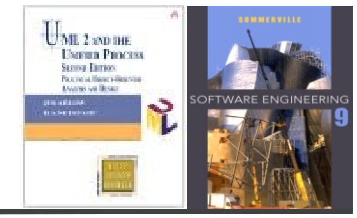
- There is a variety of different design processes that depend on the organization using the process.
- Common activities in these processes include:
  1. Define the context and modes of use of the system;
  2. Draft the system architecture;
  3. Identify the principal system processes and entities;
  4. Develop design models;
  5. Specify component/object interfaces;
  6. Finalize system architecture.

1. System context and interactions

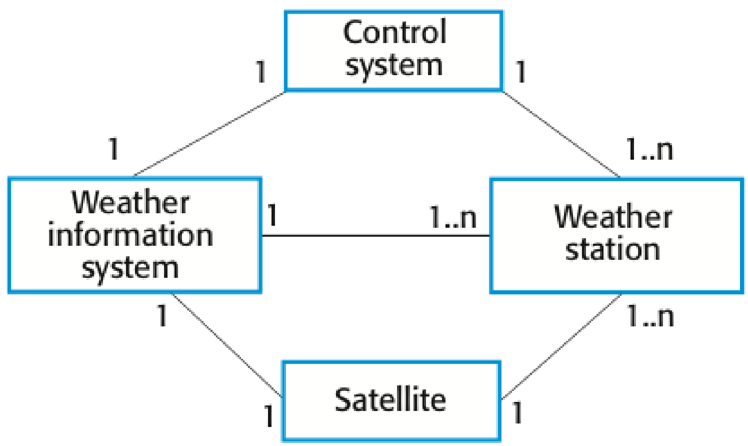


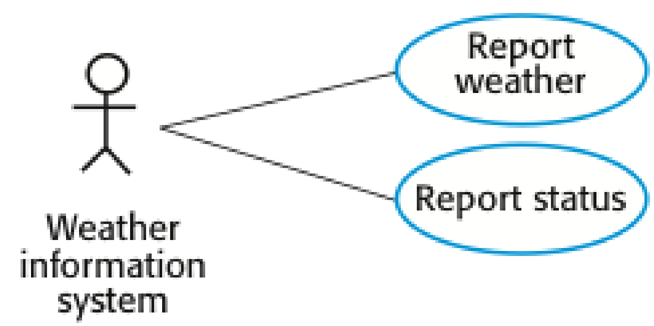
- Understanding the **relationships between the software and its external environment** is essential for deciding
  - how to provide the required system **functionality** and
  - how to **structure the system** to communicate with its environment.
- Understanding of the context also lets you establish the **boundaries** of the system.
  - Setting the system boundaries helps you decide what features are **implemented in the system** being designed and what features are **in other associated systems**.

Context and interaction models

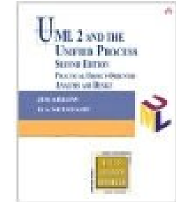


- A **system context** model is a structural model that demonstrates the users and other systems in the environment of the system being developed.
- An **interaction model** is a dynamic model that shows how the system interacts with its environment as it is used.



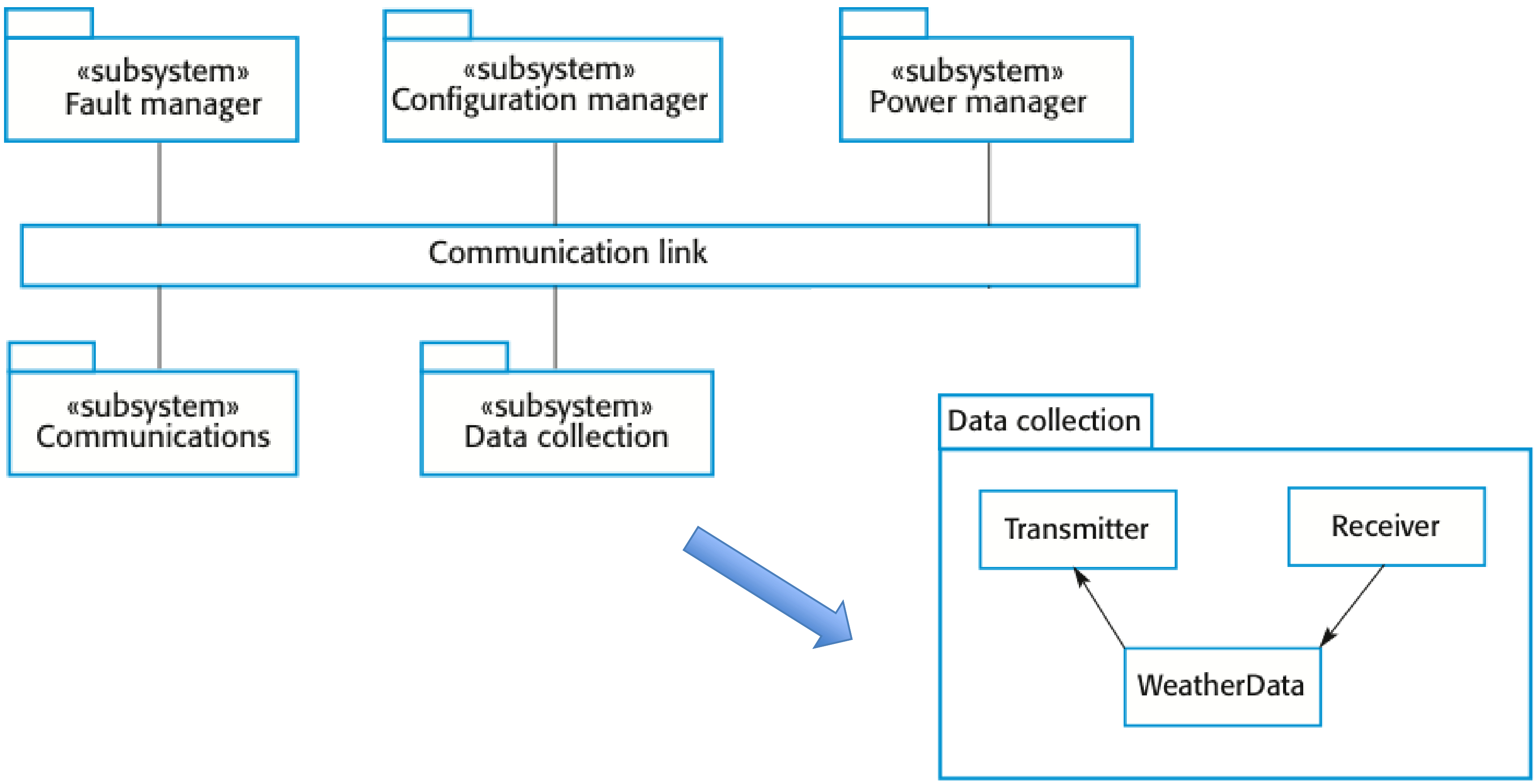


2. Architectural design

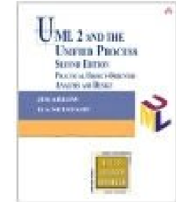


- ✧ May start system **analysis** or finish system **design**, often both.
- ✧ Represents the link between requirements specification and analysis/design processes.
- ✧ Often carried out in parallel with specification activities.
- ✧ It involves **identifying major system components and their communications**.
  - E.g. The weather station is composed of independent subsystems that communicate by broadcasting messages on a common infrastructure.

High-level architecture of the weather station

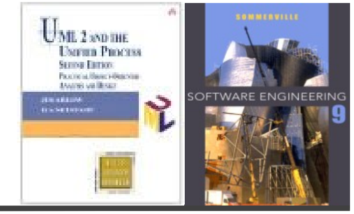


Architectural abstraction



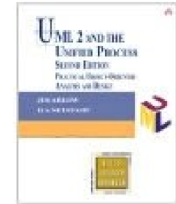
- **Architecture in the small (analysis)** is concerned with the architecture of individual programs.
  - At this level, we are concerned with the way that an individual program is decomposed into components.
- **Architecture in the large (design)** is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components.
  - These systems are distributed over different computers, which may be owned and managed by different companies.

Advantages of explicit architecture

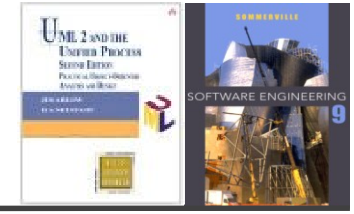


- ✧ Stakeholder communication and project planning
  - Architecture may be used to facilitate the discussion by system stakeholders.
- ✧ System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- ✧ System documentation
  - Via a complete system model that shows the different components in a system, their interfaces and their connections.
- ✧ Large-scale reuse
  - The architecture may be reusable across a range of systems
  - Product-line architectures may be developed.

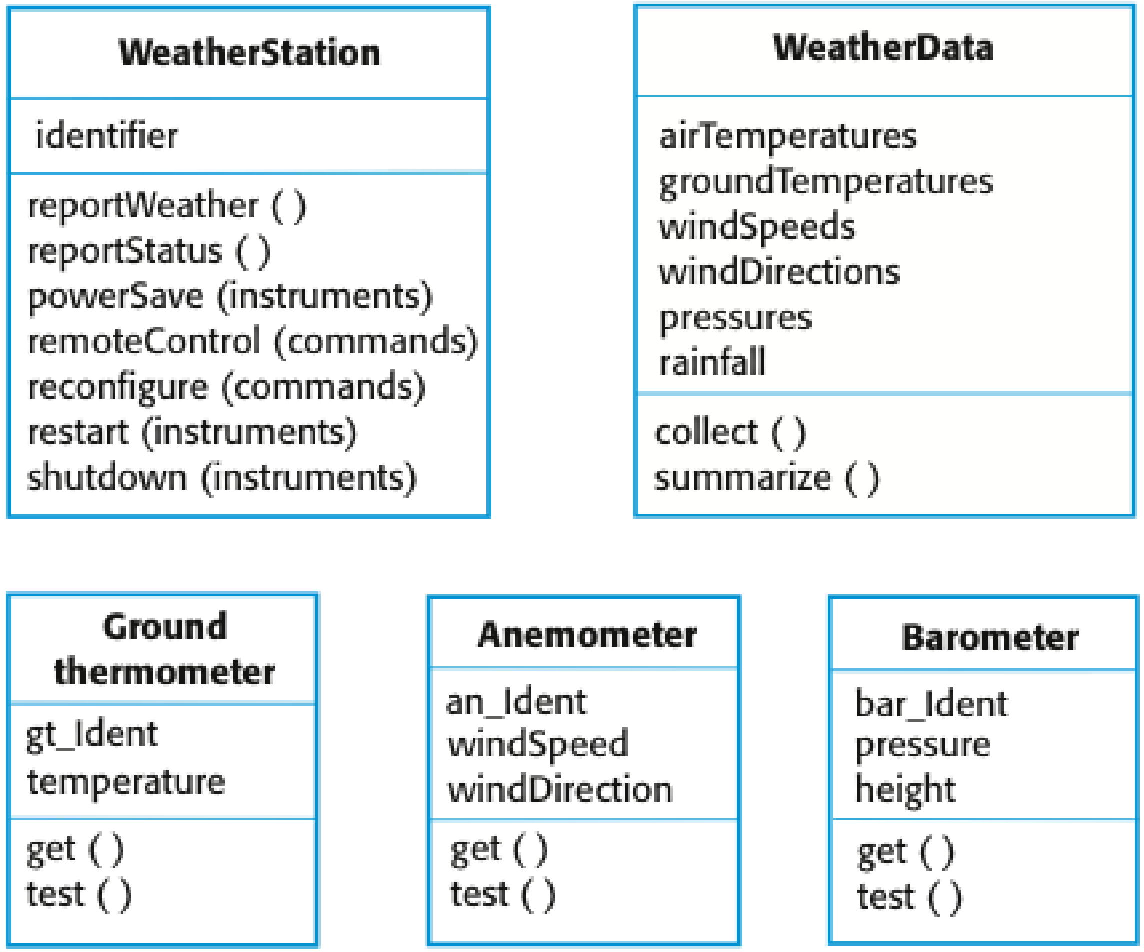
3. System analysis



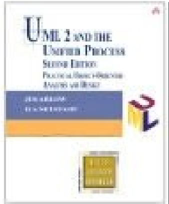
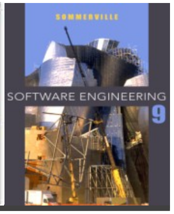
- **Identification of system entities** (object classes in object-oriented analysis) playing the key roles in the system’s problem domain, **and their relationships**.
- Distillation and documentation of key **system processes**.
- System analysis is a difficult **creative activity**.
  - There is no 'magic formula' for good analysis. It relies on the skill, experience and domain knowledge of system analysts.
- Object/relationships/processes identification is an **iterative process**. You are unlikely to get it right first time.



Weather station object classes

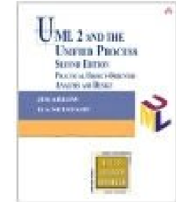


5. Design models

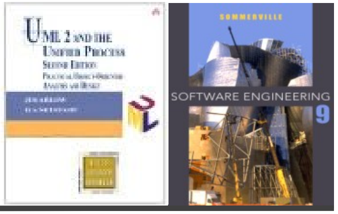



- Design models refine analysis models with the information required to **communicate and document the intended implementation** of the system.
  - E.g. Dependencies, interfaces, data-access classes, GUI classes.
- **Static models** describe the static structure of the system in terms of system entities and relationships.
  - Can you list some static UML diagrams?
- **Dynamic models** describe the dynamic interactions between entities.
  - Can you list some dynamic UML diagrams?

Key points



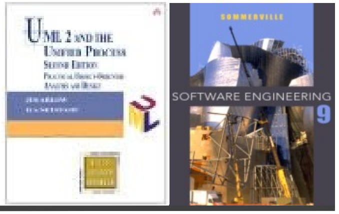
- Software analysis and design are **inter-leaved activities**. The level of detail in the design depends on the type of system and whether you are using a plan-driven or agile approach.
- The process of analysis and design includes activities to design the system architecture, identify entities in the system, describe the design using different models and document the component interfaces.
- **Software analysis** is a creative activity in which you identify software processes, entities (objects) and their relationships.
- **Software design** refines analytical models with implementation details.



Structured vs. Object-Oriented Methods

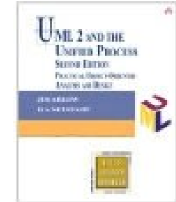
Lecture 4/Part 2

Fundamental views of software systems



- Function oriented view
  - System as a set of interacting functions. Functional transformations based in processes, interconnected with data and control flows.
- Data oriented view
  - Searches for fundamental data structures in the system. Functional aspect of the system (i.e. data transformation) is less significant.
- Object oriented view
  - System as a set of interacting objects, encapsulating both the data and operations performed on the data.

Structured vs. object-oriented analysis



✧ Structured analysis

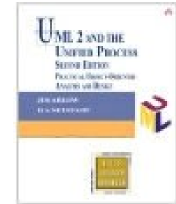
- Driven by the **function oriented view**, in synergy with **data oriented view**, through the concept of functional decomposition.

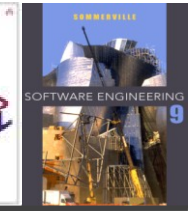
✧ Object-oriented analysis

- Driven by the **object oriented view**.

✧ Do they have anything in common?

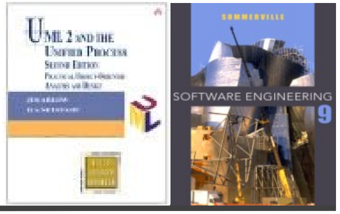
Structured analysis and design

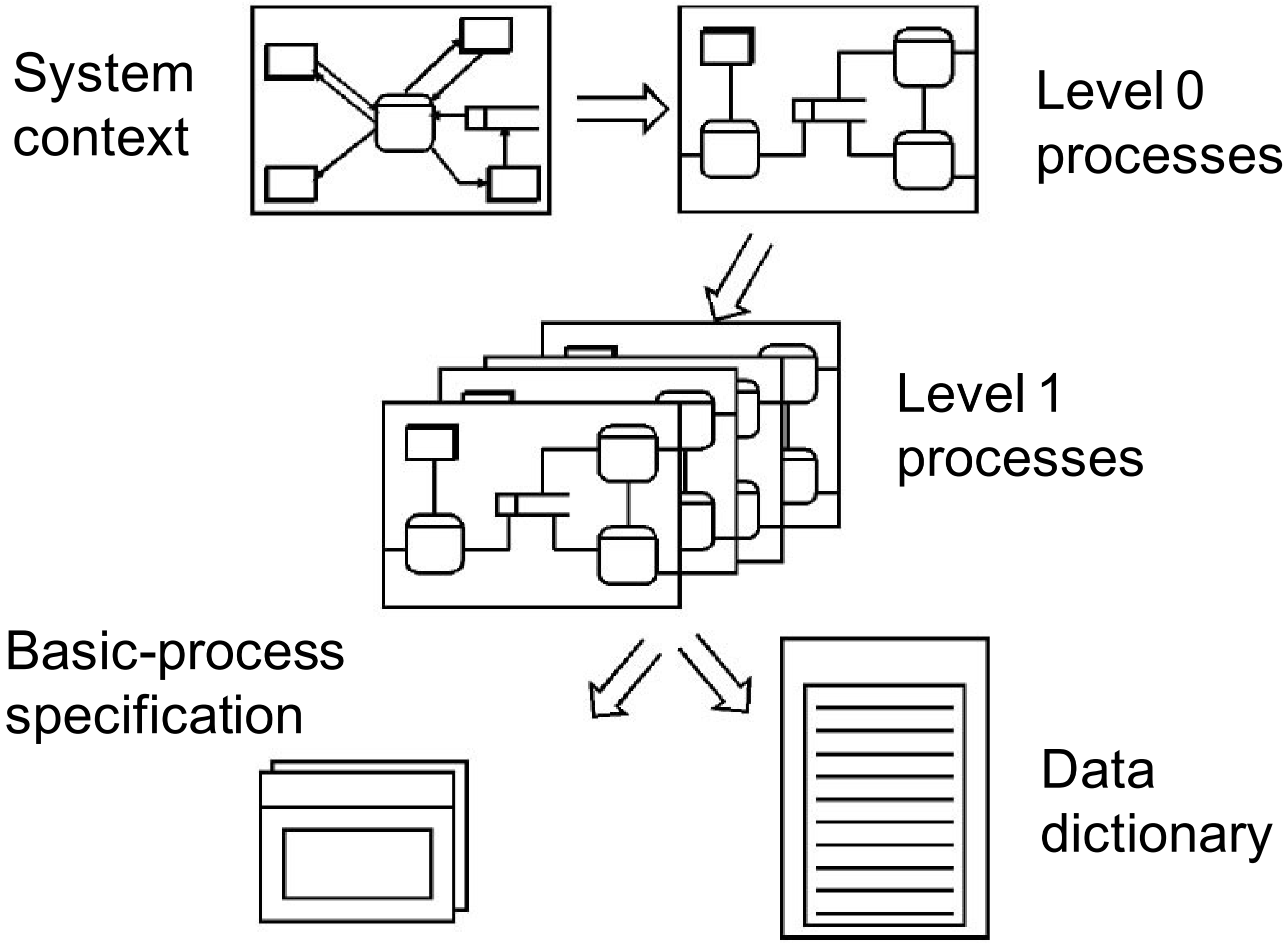




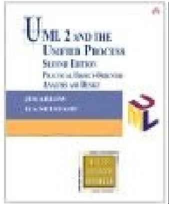
- Divides a project on small, well defined activities and defines the order and interaction of the activities.
- Using **hierarchical graphical techniques**, resulting in a detailed structured specification, which can be understood by both system engineers and users.
- Effective in project **structuring to smaller parts**, which simplifies time and effort estimates, deliverables control and project management as such.
- Aimed at increasing system quality.

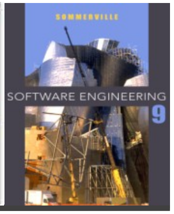
Functional decomposition





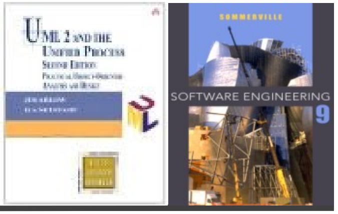
Structured methods





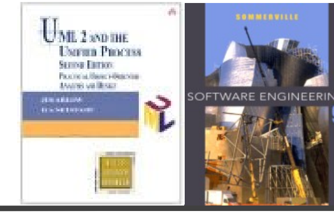
- ✧ DeMarco: Structured Analysis and System Specification (SASS)
- ✧ Gane-Sarson: Logical Modelling (LM)
- ✧ **Yourdon: Modern Structured Analysis (YMSA)**
  - Concentrates on the data and control flow of system processes and sub-processes.
- ✧ **Structured Systems Analysis and Design Method (SSADM)**
  - Physical design, logical process design and logical data design

Core notations of structured methods



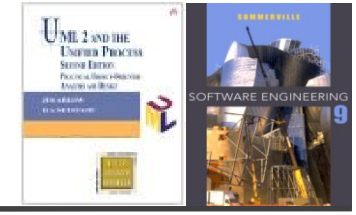
- ✧ Context diagram
  - Models system boundary and environment.
- ✧ Data flow diagram (DFD)
  - Models the system as a network of processes completing designated functions and accessing system data.
- ✧ Entity relationship diagram (ERD)
  - Models system's data.
- ✧ State diagram (STD)
  - Models system states and actions guarding transitions from one state to another.

Examplary method (Gane-Sarson)



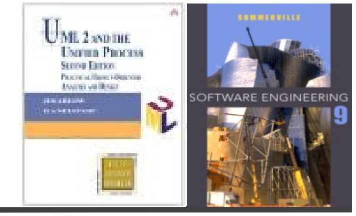
1. Define system context and create initial system DFD.
2. Draft initial data model (ERD).
3. Analyze data entities and relationships into final ERD.
4. Refine DFD according to the ERD data model (create logical process model).
5. Decompose logical process model into procedural elements.
6. Specify the details of each individual procedural element.

Object-oriented analysis and design



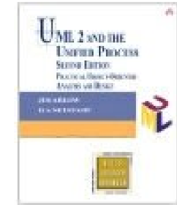
- Software engineering approach that models a system as a group of **interacting objects**.
- Each **object** represents some entity of interest in the system being modeled, and is characterized by its **class**, its **state** (data elements), and its **behavior**.
- **Various models** can be created to show the static structure, dynamic behavior, and run-time deployment of these collaborating objects.
- There is a number of **different methods**, defining the ordering of modeling activities. The **modeling notation** uses to be unified (UML).

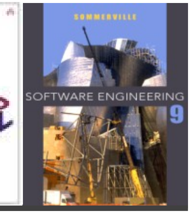
Object-oriented methods



- ✧ Jim Rumbaugh: Object Modelling Technique (OMT)
- ✧ Coad-Yourdon: Method for Object-Oriented Analysis (OOA)
- ✧ Jacobson: Object-Oriented Software Engineering (OOSE)
- ✧ **Kruchten et al.: Rational Unified Process (RUP)**
  - Risk-driven iterations, component-based, with continuous quality verification and change management.
- ✧ **Booch-Jacobson-Rumbaugh: Unified Process (UP)**
  - Simplified non-commercial version of RUP maintained by Object Management Group (OMG).

UML notation for object-oriented methods





✧ External perspective

- **Use case diagram**

✧ Structural perspective

- **Class diagram**, Object diagram, Component diagram, Package diagram, Deployment diagram, Composite structure diagram

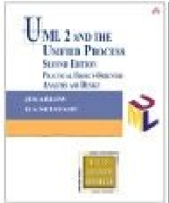
✧ Interaction perspective

- **Sequence diagram**, Communication diagram, Interaction overview diagram, Timing diagram

✧ Behavioral perspective

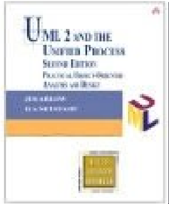
- **Activity diagram**, State diagram

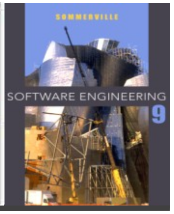
Examplary method (Unified Process, analysis and design excerpt)




1. Requirements
   - System boundary, actors and requirements modelling with **Use Case diagram**.
2. Analysis
   - Identification of analysis classes, relationships, inheritance and polymorphism, and their documentation with a **Class diagram**.
   - Use Case realization with **Interaction** and **Activity diagrams**.
3. Design
   - Design classes, interfaces and components, resulting in refined **Class diagrams** and **Component diagrams**.
   - Detailed Use Case realization with **Interaction** and **State diagrams**.

Key points





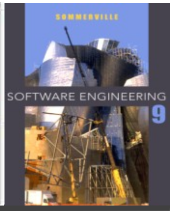
✧ Structured methods

- System as a set of nested processes accessing system data.

✧ Object-oriented methods

- System as a set of interacting objects (functions and data).

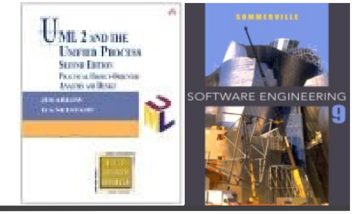
	Structured analysis	Object-oriented analysis
System boundary	Context diagram	Use case diagram
Functionality	Data flow diagram	Activity diagram Interaction diagrams
Data	Entity-relationship diagram	Class and Object diagram
Control	State diagram	State diagram

Object-Oriented Analysis in UML

Lecture 4/Part 3

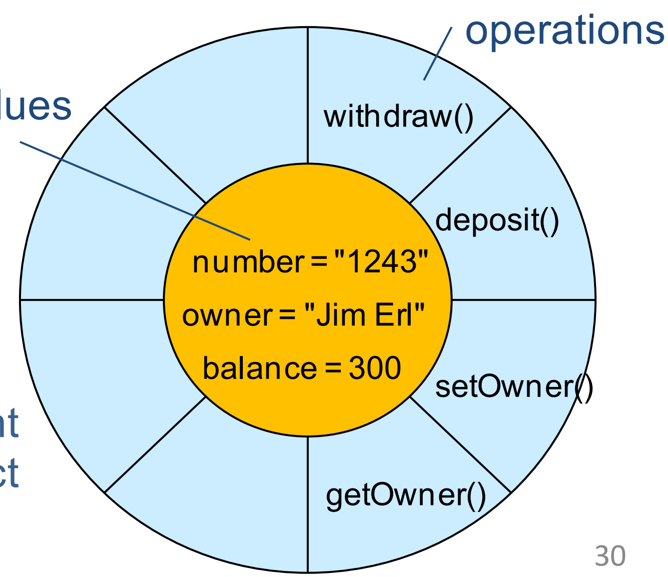
Analysis objects and classes



What are objects?

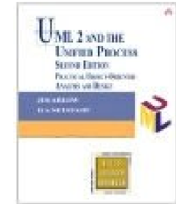
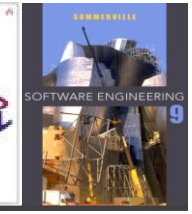
- Objects consist of data and function packaged together in a reusable unit. Objects **encapsulate** data.
- Every object is an instance of some **class** which defines the common set of **features** (attributes and operations) shared by all of its instances.
- Objects have:
  - **Attribute values** – the data part
  - **Operations** – the behaviour part

attribute values



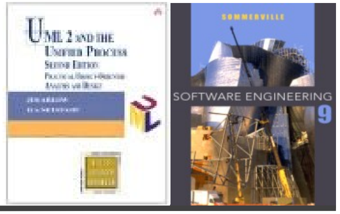
Bank Account
Object

All objects have

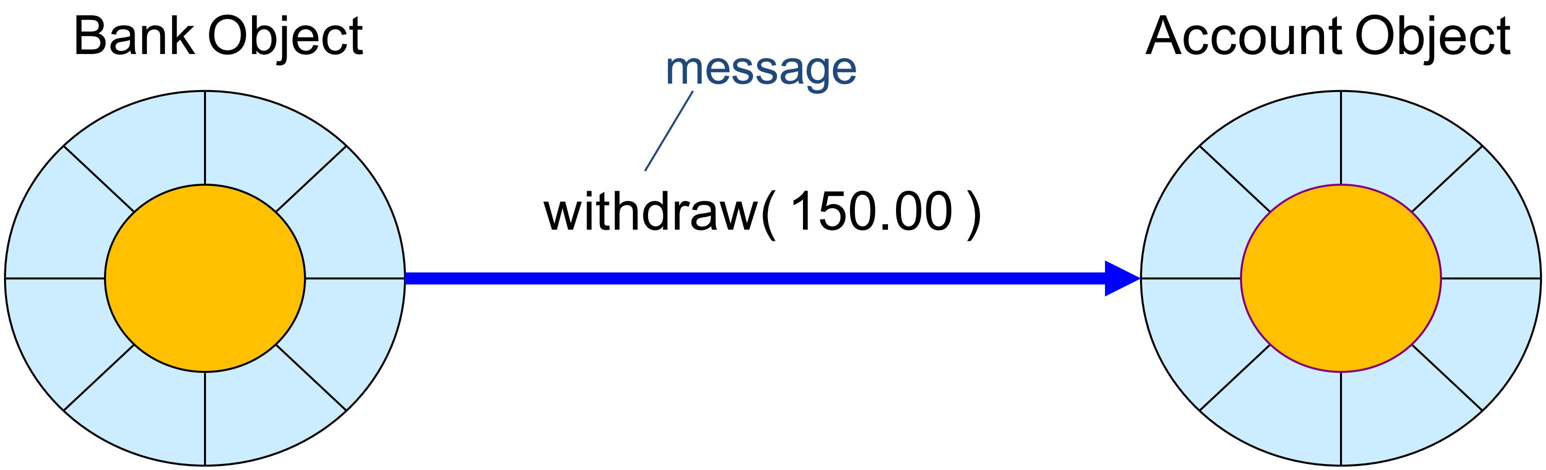



- **Identity:** Each object has its own unique identity and can be accessed by a unique handle
  - Distinguish two cars of the same type and one car referenced from two places.
- **State:** This is the actual data values stored in an object at any point in time
  - On and off for a light bulb (one attribute).
  - On + busy, on + idle, off for a printer (two attributes).
- **Behaviour:** The set of operations that an object can perform

Messaging



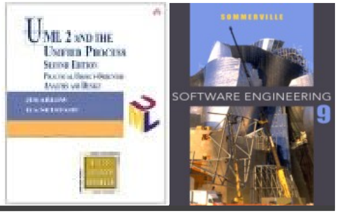
- In OO systems, objects send messages to each other over links
- These messages cause an object to invoke an operation



the Bank object sends the message “withdraw 150.00” to an Account object.

the Account object responds by invoking its withdraw operation. This operation decrements the account balance by 150.00.

UML Object Syntax



object identifier
(*must* be underlined)

object name

class name

name compartment

attribute compartment

jimsAccount : Account

accountNumber : String = "1234567"
owner : String = "Jim Arlow"
balance : double = 300.00

attribute name

attribute type

attribute value

variants
(N.B. we've omitted the attribute compartment)

object and class name

jimsAccount : Account

object name only

jimsAccount

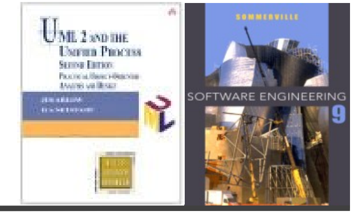
class name only

: Account

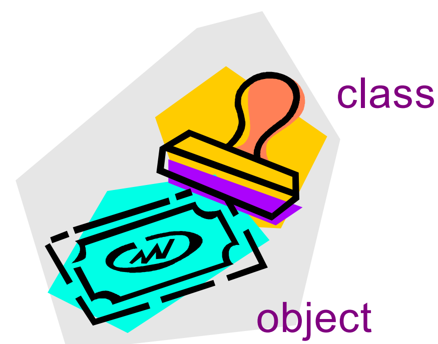
an anonymous object

- All objects of a particular class have the same set of operations. They are not shown on the object diagram, they are shown on the class diagram (see later)
- Attribute types are often omitted to simplify the diagram
- Naming: object and attribute names in lowerCamelCase, class names in UpperCamelCase

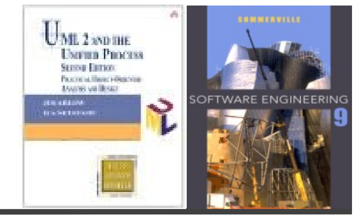
What are classes?



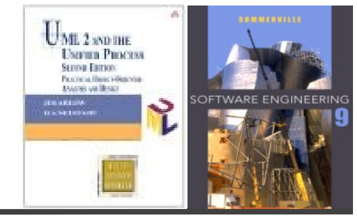
- Every object is an instance of one class - the class describes the "type" of the object
- Classes allow us to model sets of objects that have the same set of features - **a class acts as a template for objects**:
  - The class determines the structure (set of features) of all objects of that class
  - All objects of a class **must** have the same set of operations, **must** have the same attributes, but **may** have different attribute values
- **Classification** is one of the most important ways we have of organising our view of the world
- Think of classes as being like:
  - Rubber stamps
  - Cookie cutters



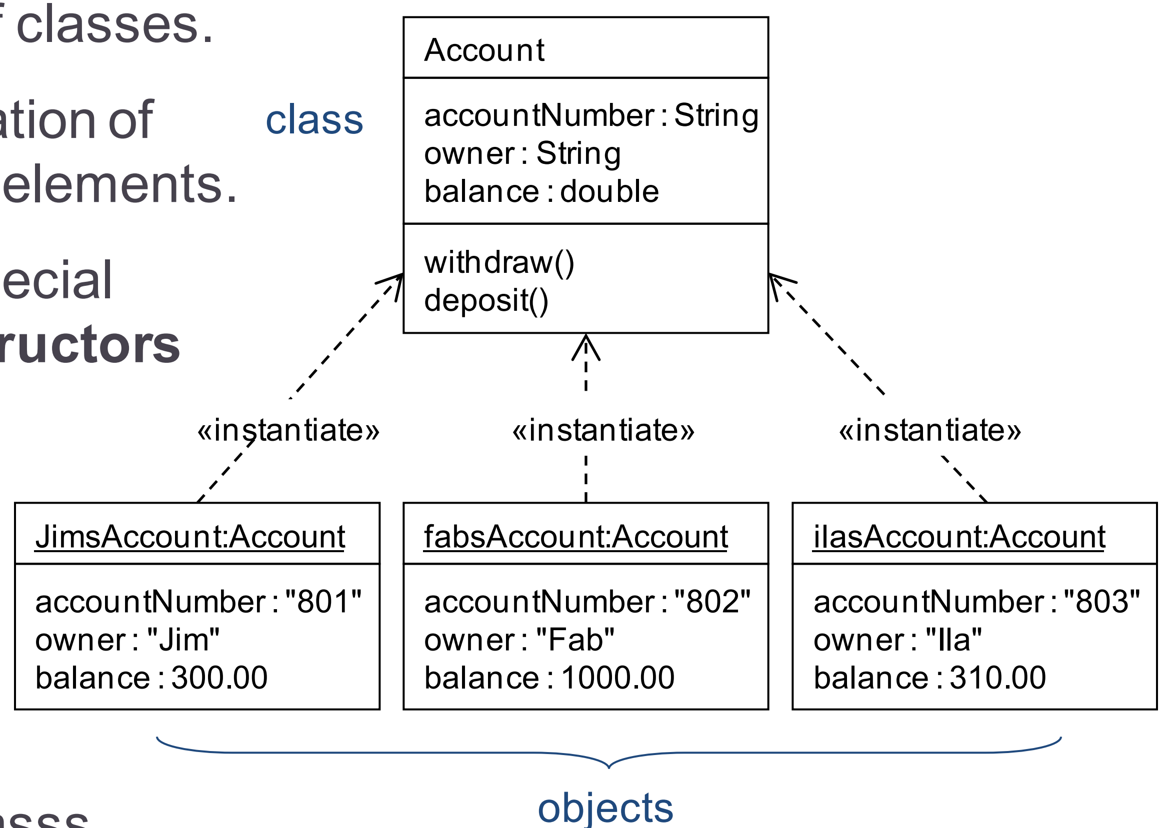
Exercise - how many classes?



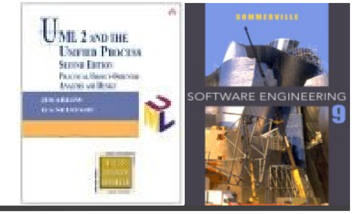
Classes and objects

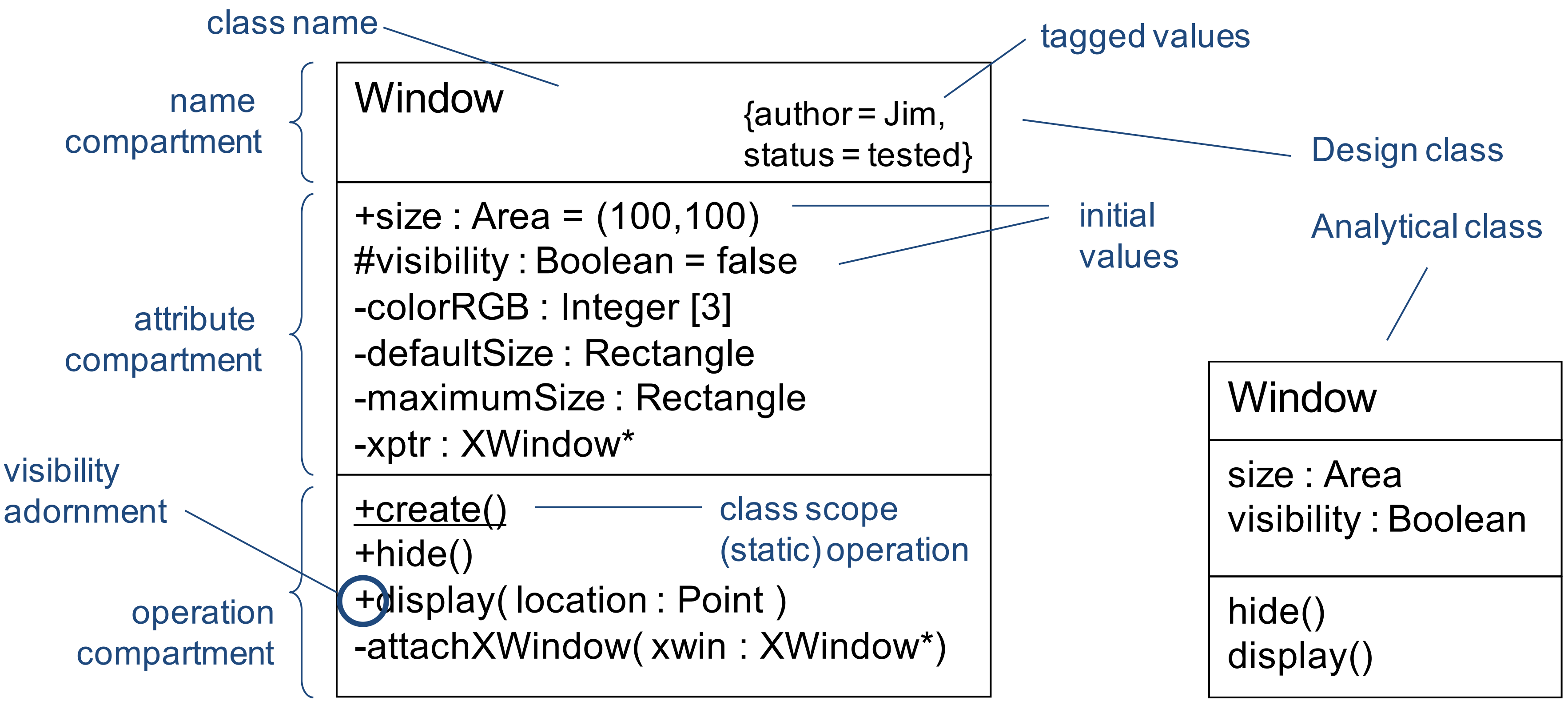


- Objects are instances of classes.
- **Instantiation** is the creation of new instances of model elements.
- Most classes provide special operations called **constructors** to create instances of that class.
- These operations have **class-scope** i.e. they belong to the class itself rather than to objects of the classs.



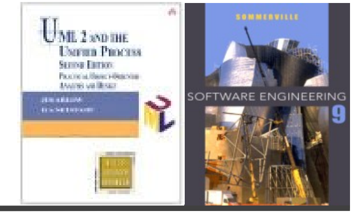
UML class notation





- Classes are named in UpperCamelCase – **avoid abbreviations**!
- Use descriptive names that are nouns or noun phrases

Attribute compartment



Structure

visibility name : type multiplicity = initialValue

mandatory

Visibility

- \+ public
- \- private
- \# protected
- ~ package

Type

Integer, Real, Boolean, String, Class

Multiplicity

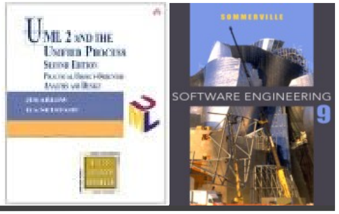
- [3] specific number of elements
- [0..1] optional
- * array, list

Initial values

attribute compartment

```
Window                         {author = Jim,
                                status = tested}
------------------------------------------------
+size : Area = (100,100)
#visibility : Boolean = false
-colorRGB : Integer [3]
-defaultSize : Rectangle
-maximumSize : Rectangle
-xptr : XWindow*
------------------------------------------------
+create()
+hide()
+display( location : Point )
-attachXWindow( xwin : XWindow*)
```

Operation compartment



Operation signature

visibility name (direction parameterName : parameterType = default, ...) : returnType

parameter list

or a list r1, r2,… rn

Direction

in	input value, default
out	repository for system output
inout	modifiable input value
return	operation return value(s)

Scope

instance scope	defaults
class scope	underlined

Constructors

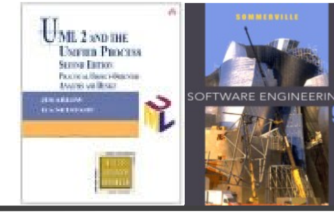
generic constructor name or
Java/C++ standard

+BankAccount(aNumber : int)

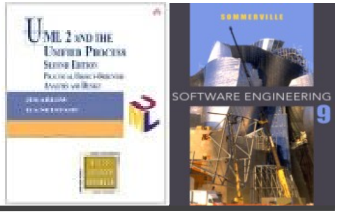
operation compartment

BankAccount
-accountNumber : int -count : int = 0
+create(aNumber : int) +getNumber() : int -incrementCount() +getCount() : int

Key points



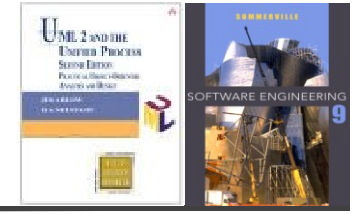
- We have looked at objects and classes and examined the relationship between them
- We have explored the UML syntax for modelling classes including:
  - Attributes
  - Operations
- We have seen that scope controls access
  - Class scope attributes are shared by all objects of the class and are useful as counters
  - Attributes and operations are normally instance scope
  - We can use class scope operations for constructor and destructors



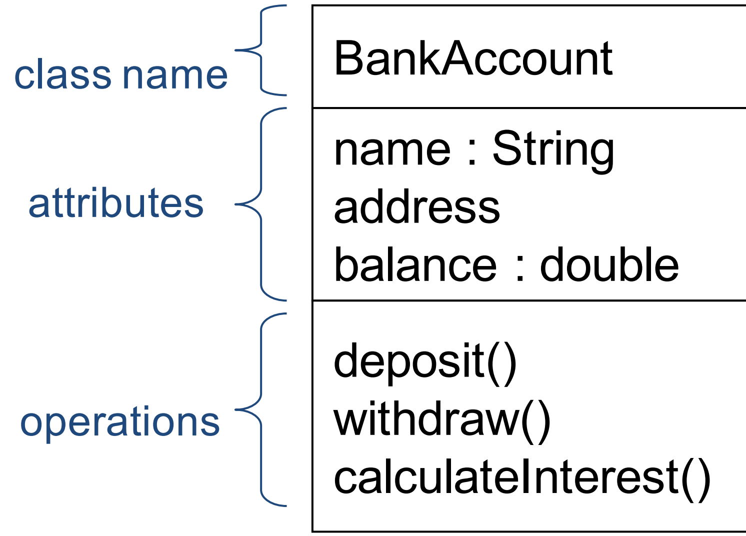
Finding Analysis Classes

Lecture 4/Part 4

What are Analysis classes?

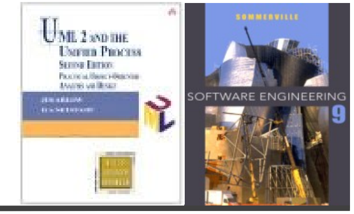


- Analysis classes represent a crisp abstraction in the problem domain
  - They may ultimately be refined into one or more design classes
- Analysis classes have:
  - A very "high level" set of attributes. They indicate the attributes that the design classes might have.
  - Operations that specify at a high level the key services that the class must offer. In Design, they will become actual, implementable, operations.



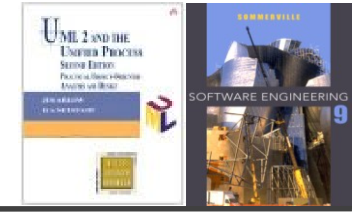
Specify attribute types if you know what they are.

What makes a good analysis class?



- Its name reflects its intent
- It is a **crisp abstraction** that models one specific element of the problem domain
  - It maps onto a clearly identifiable feature of the problem domain
- It has **high cohesion**
  - Cohesion is the degree to which a class models a single abstraction
  - Cohesion is the degree to which the responsibilities of the class are semantically related
- It has **low coupling**
  - Coupling is the degree to which one class depends on others

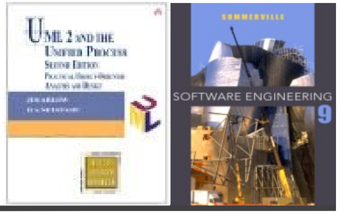
Rules of thumb



- ✧ 3 to 5 responsibilities per class
- ✧ Each class collaborates with others
- ✧ Beware many very small classes
- ✧ Beware few but very large classes
- ✧ Beware of “functoids”
- ✧ Beware of “omnipotent” classes
- ✧ Avoid deep inheritance trees

A *responsibility* is a contract or obligation of a class - it resolves into operations and attributes

Finding classes



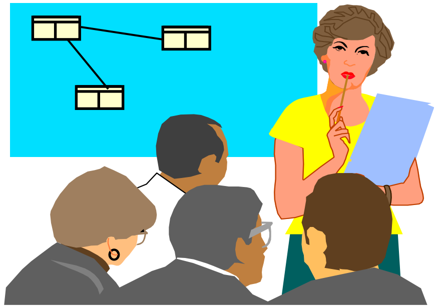
- Perform noun/verb analysis on documents:
  - Nouns are candidate **classes**
  - Verbs are candidate **responsibilities**

  What documents can be studied?
- Perform CRC card analysis
  - Class, Responsibilities and Collaborators
  - A two phase brainstorming technique using sticky notes – first brainstorm and then analyse the dat

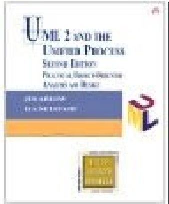
things the class does

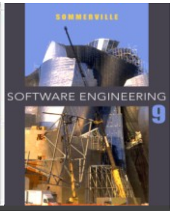
Class Name: BankAccount	
Responsibilities:	Collaborators:
Maintain balance	Bank

things the class works with



Other sources of classes





- Physical objects
- Paperwork, forms
  - Be careful when relying on processes that need to change
- Known interfaces to the outside world
- Conceptual entities that form a cohesive abstraction

- **With all techniques, beware of spurious classes**
  - Look for **synonyms** - different words that mean the same
  - Look for **homonyms** - the same word meaning different things

Key points


- We've looked at what constitutes a well-formed analysis class
- We have looked at two analysis techniques for finding analysis classes:
  - Noun verb analysis of use cases, requirements, glossary and other relevant documentation
  - CRC analysis