CS 3300
Intro to Software Engineering

Software Design

Mahdi Roozbahani

Slides are based on Alex Orso.

Software design

- System organization that satisfies functional and non-functional requirements
- Input:
  - Specification – what to do
- Output:
  - Design document – how to do it

Why design?

- good design ➡ good code
- easier to code, test, maintain, change
- easier to understand impact of requirements changes
- large projects – divide across teams, but have unifying design

Overview of Design Phase

- Main activities
  - Architectural (high-level) design
    - Decompose the product into modules
    - Identify connections between them
  - Detailed (low-level) design
    - Choose data structures
    - Select/design algorithms
  - UI design (if applicable)
  - Design testing
    - Make sure design is correct – performed throughout phase
- Two key aspects of product for design: actions & data
  => action-oriented, data-oriented, or hybrid

Design: art or science?

- Design is the most creative phase
  - No set rules that you must follow
  - A lot of its success boils down to experience
  - but there are principles and patterns which improve quality of design

Architectural design

Architectural design

- Process of identifying and assigning the responsibility for aspects of functional behavior to various modules or components of a software system
- The communication interfaces among the components must also be specified

https://www.youtube.com/watch?time_continue=1&v=sb7y8ReF_IQ&feature=emb_logo

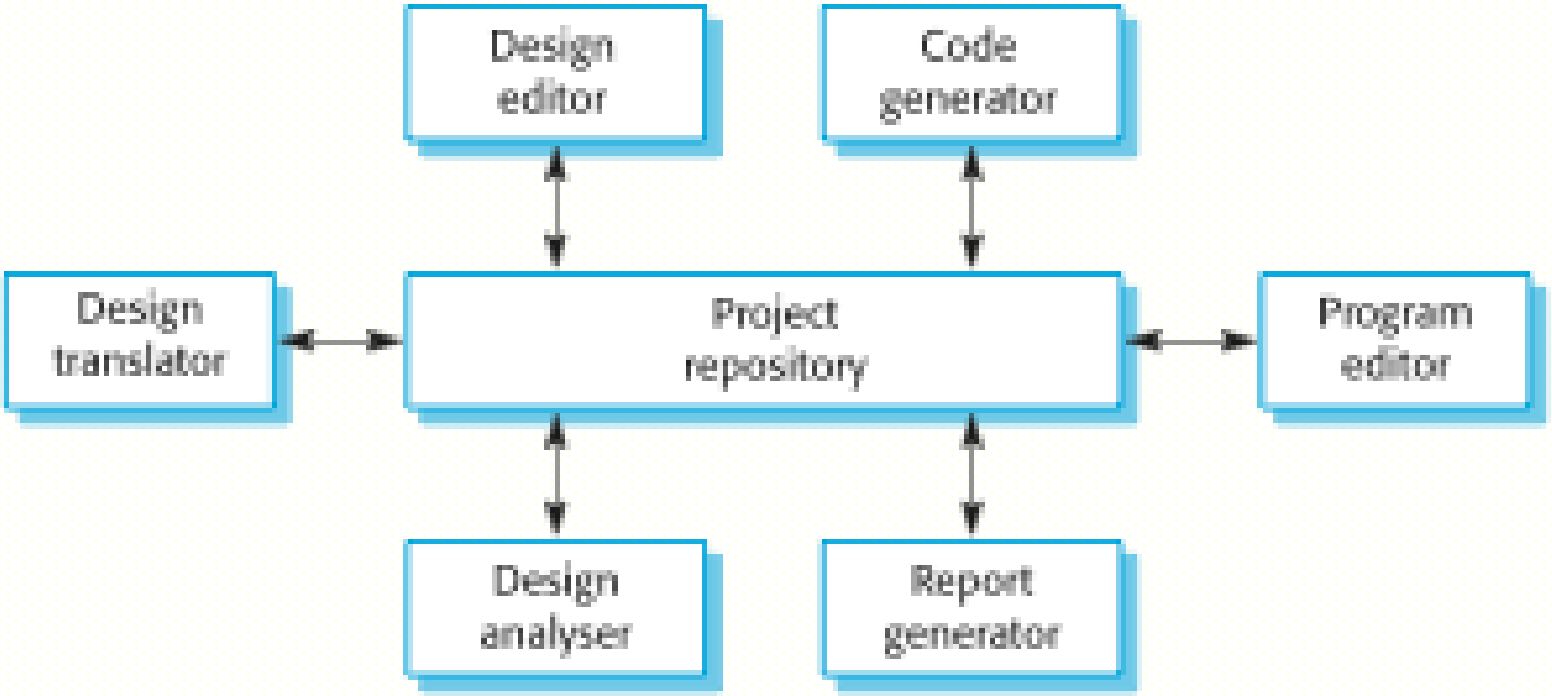
System architecture

- Reflects the basic strategy that is used to structure a system.
- Three organizational styles are widely used:
  - A shared data repository style;
  - A shared services and servers style;
  - An abstract machine or layered style.

The repository model

- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central DB or repository and may be accessed by all sub-systems;
  - Each sub-system maintains its own DB and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared, the repository model of sharing is most commonly used.

E.g., CASE toolset architecture or Email



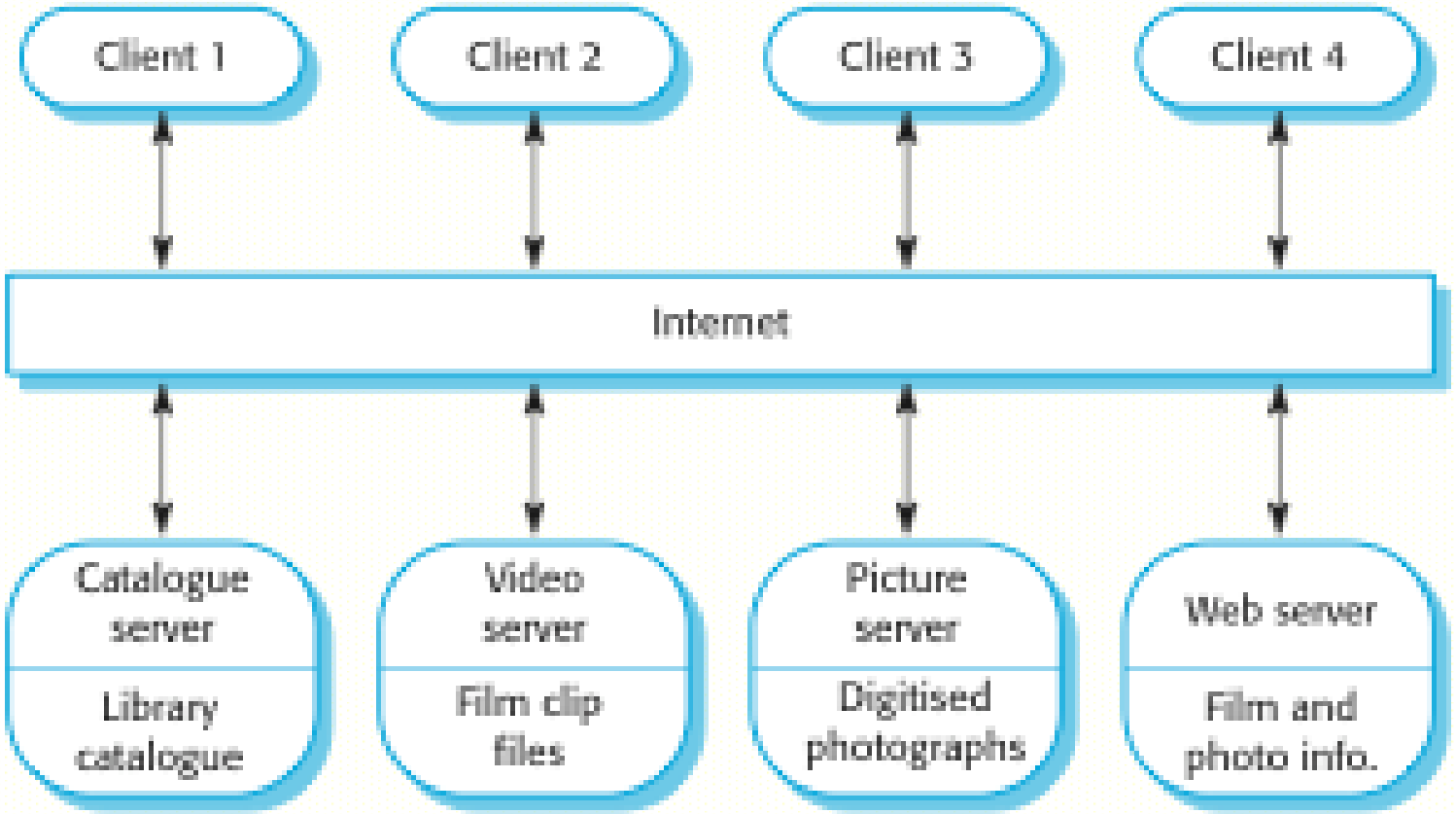
Repository model characteristics

- Advantages
  - Efficient way to share large amounts of data;
  - Sub-systems need not be concerned with how data is managed (centralised management of backup, security, etc.)
  - Sharing model is published as the repository schema
    ➡ easy integration
- Disadvantages
  - Sub-systems must agree on a repository data model ➡ compromise;
  - Data evolution is difficult and expensive;
  - No scope for specific management policies;

Client-server model

- Distributed system model which shows how data and processing is distributed across a range of components.
- Set of stand-alone servers that provide specific services such as printing, data management, etc.
- Set of clients which call on these services.
- Network that allows clients to access servers.

E.g., film and picture library



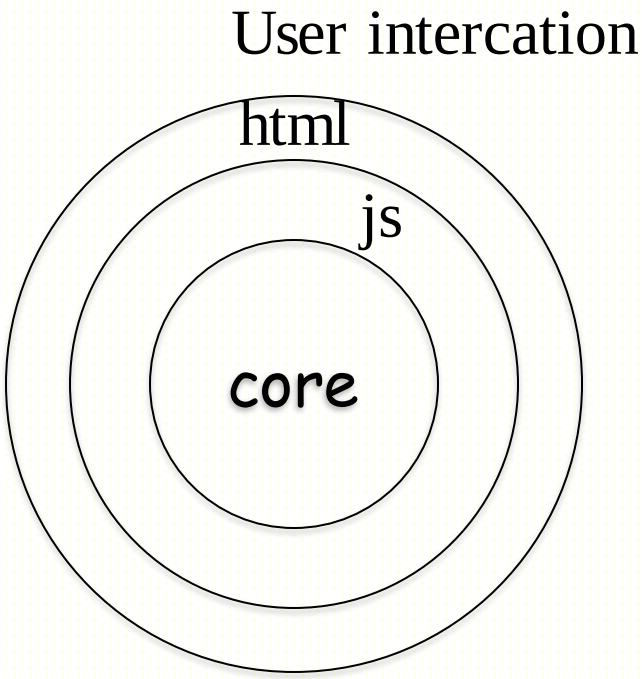
Client-server characteristics

- Advantages
  - Distribution of data is straightforward;
  - Makes effective use of networked systems. May require cheaper hardware;
  - Highly decoupled;
  - Easy to add new servers or upgrade existing servers.
- Disadvantages
  - No shared data model, so sub-systems use different data organisation. Data interchange may be inefficient;
  - Redundant management in each server;
  - No central register of names and services – it may be hard to find out what servers and services are available.

Layered model

- Used to model the interfacing of sub-systems.
- Organises the system into a set of layers (or abstract machines) each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.

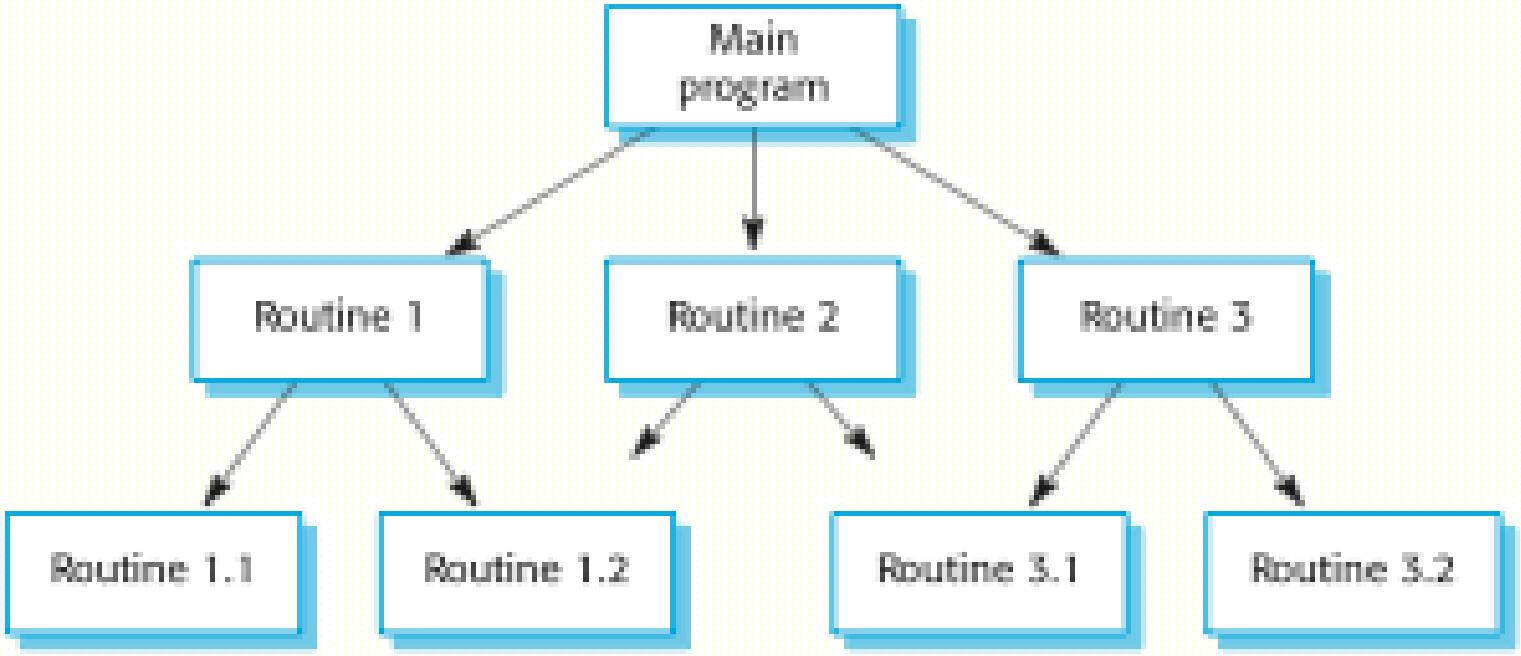
E.g., Web Application



Control styles

- Are concerned with the control flow between sub-systems. Distinct from the system decomposition model.
- Centralized control
  - One sub-system has overall responsibility for control and starts and stops other sub-systems.
- Event-based control
  - Each sub-system can respond to externally generated events from other sub-systems or the system’s environment.

Centralized call-return model

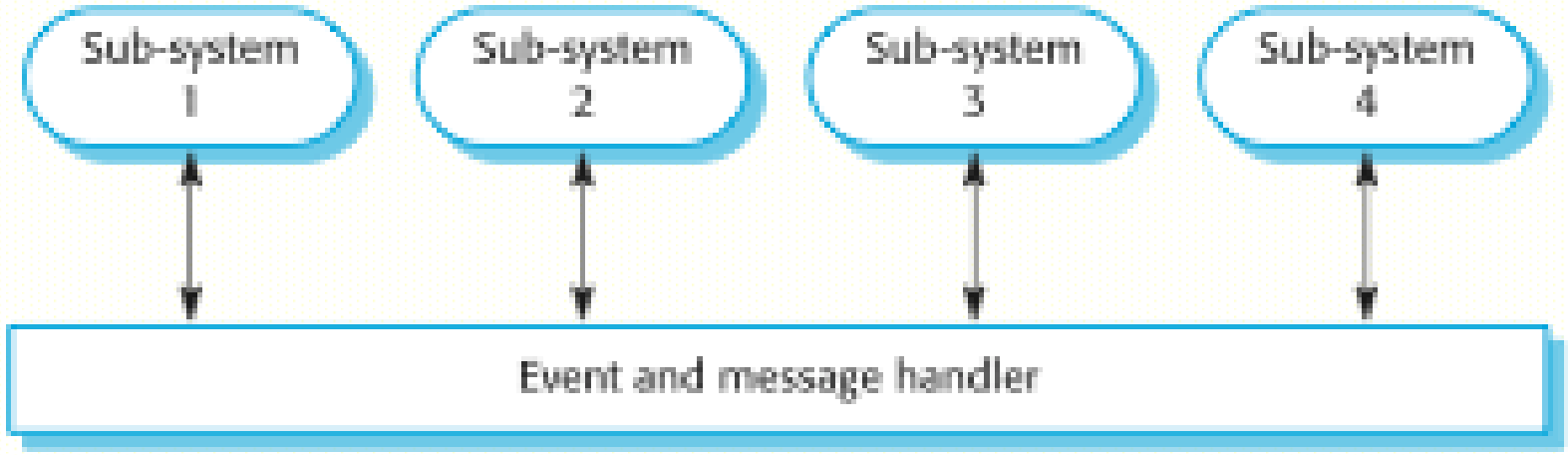


https://www.ijser.org/researchpaper/Control-Models-in-Software-Engineering.pdf

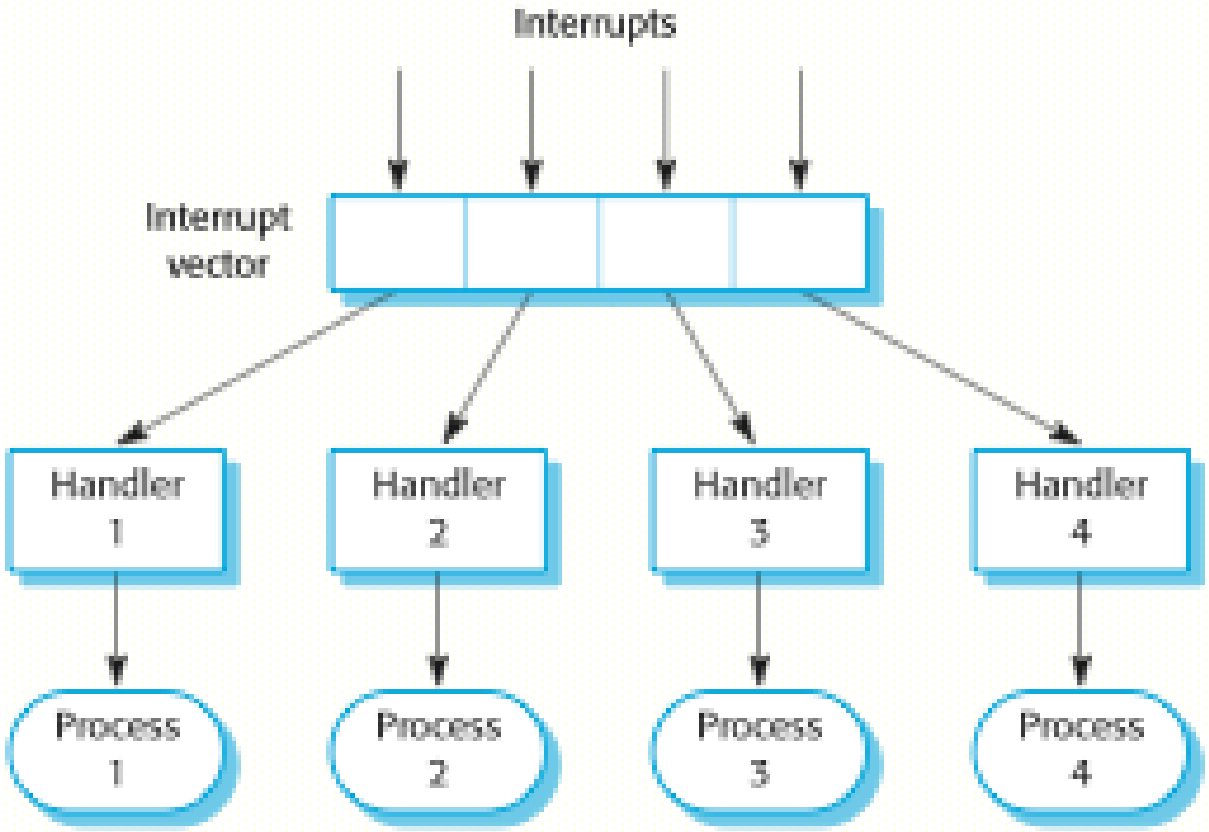
Event-driven systems

- Driven by externally generated events where the timing of the event is out with the control of the sub-systems which process the event.
- Two principal event-driven models
  - Broadcast models. An event is broadcast to all sub-systems. Any sub-system which can handle the event may do so;
  - Interrupt-driven models. Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing.

Broadcasting



Interrupt-driven control



Detailed design

Detailed design

Detailed design is the process of specifying the logical behavior of each component

- Algorithm selection
- Data structure representation
- Combination of natural language, pseudo code, graphical representation

Design models

- Different design models may be produced during the design process
- Each model presents different perspectives on the design

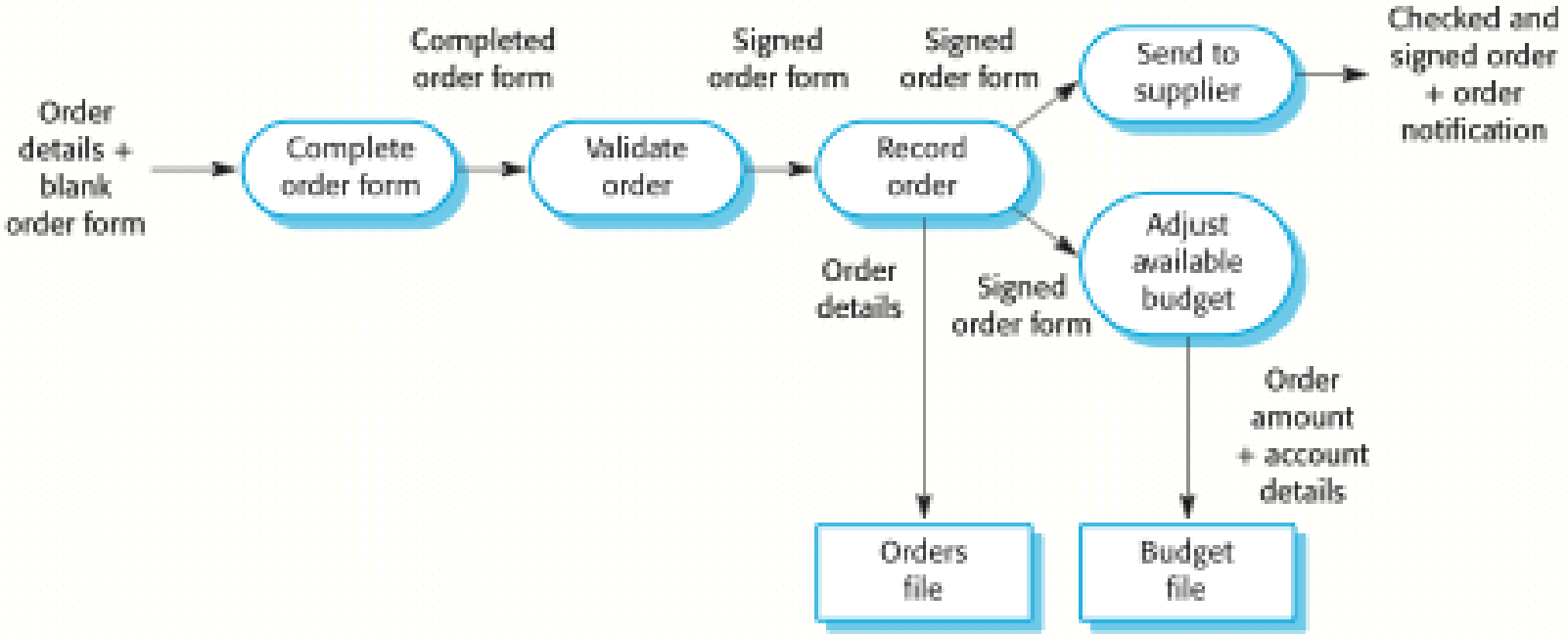
Behavioural models

- Behavioural models are used to describe the overall behaviour of a system.
- Two types of behavioural model are:
  - Data processing models that show how data is processed as it moves through the system;
  - State machine models that show the systems response to events.
- These models show different perspectives so both of them may be needed to describe the system's behaviour.

Data-processing models

- Data flow diagrams (DFDs) may be used to model the system's data processing.
- These show the processing steps as data flows through a system.
- Simple and intuitive notation that customers can understand.
- Show end-to-end processing of data.

E.g., order processing DFD



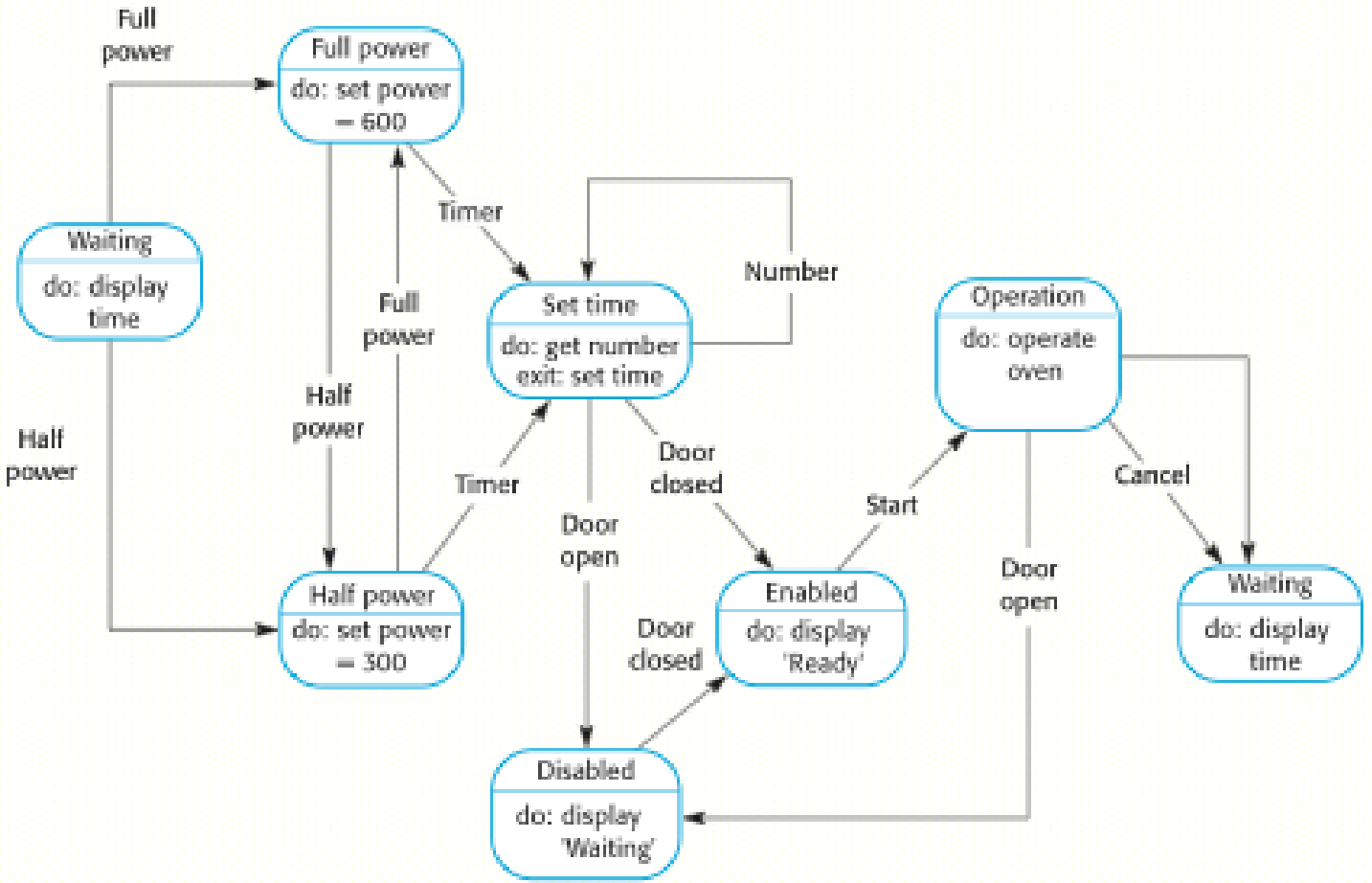
State machine models

- Model the behavior of the system in response to external and internal events.
- They show the system's responses to stimuli.
- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.
- Statecharts are an integral part of the UML and are used to represent state machine models.

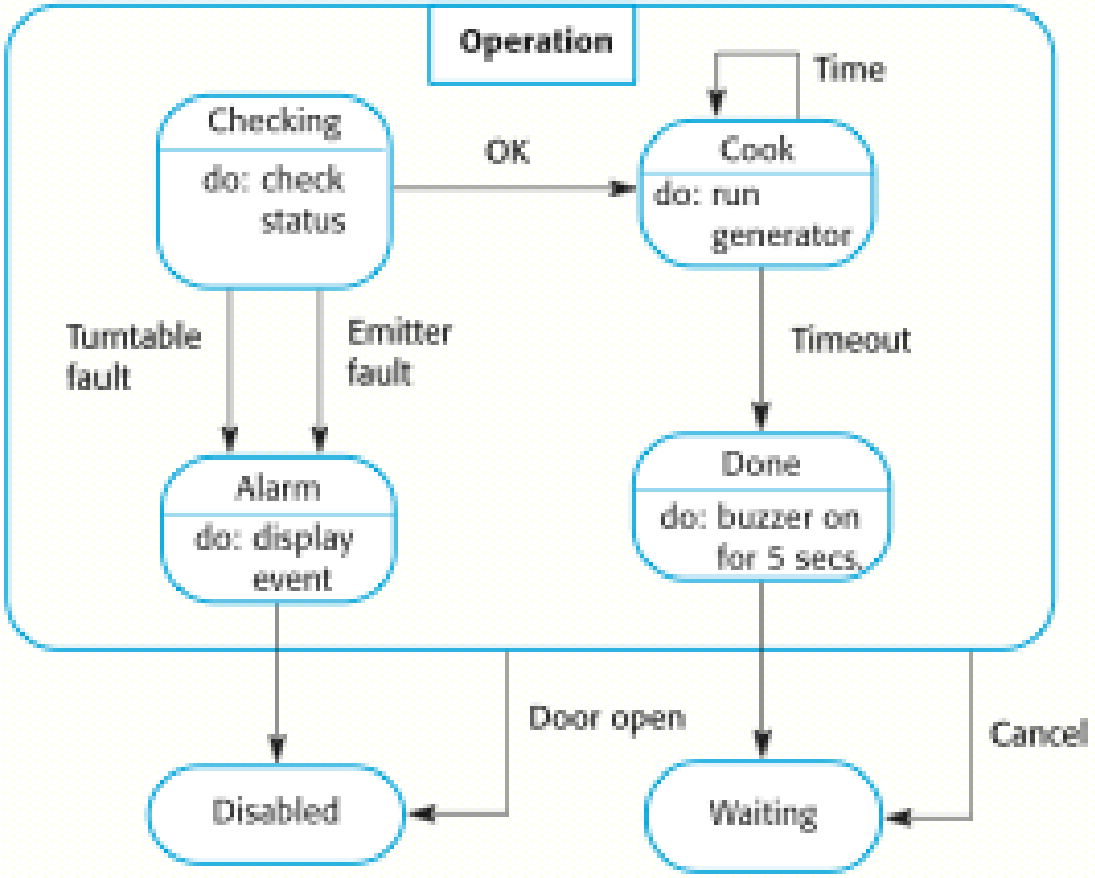
Statecharts

- Allow for decomposing a model into sub-models (see following slide).
- A brief description of the actions is included following the ‘do’ in each state.
- Can be complemented by tables describing the states and the stimuli.

E.g., microwave oven model



E.g., microwave oven operation



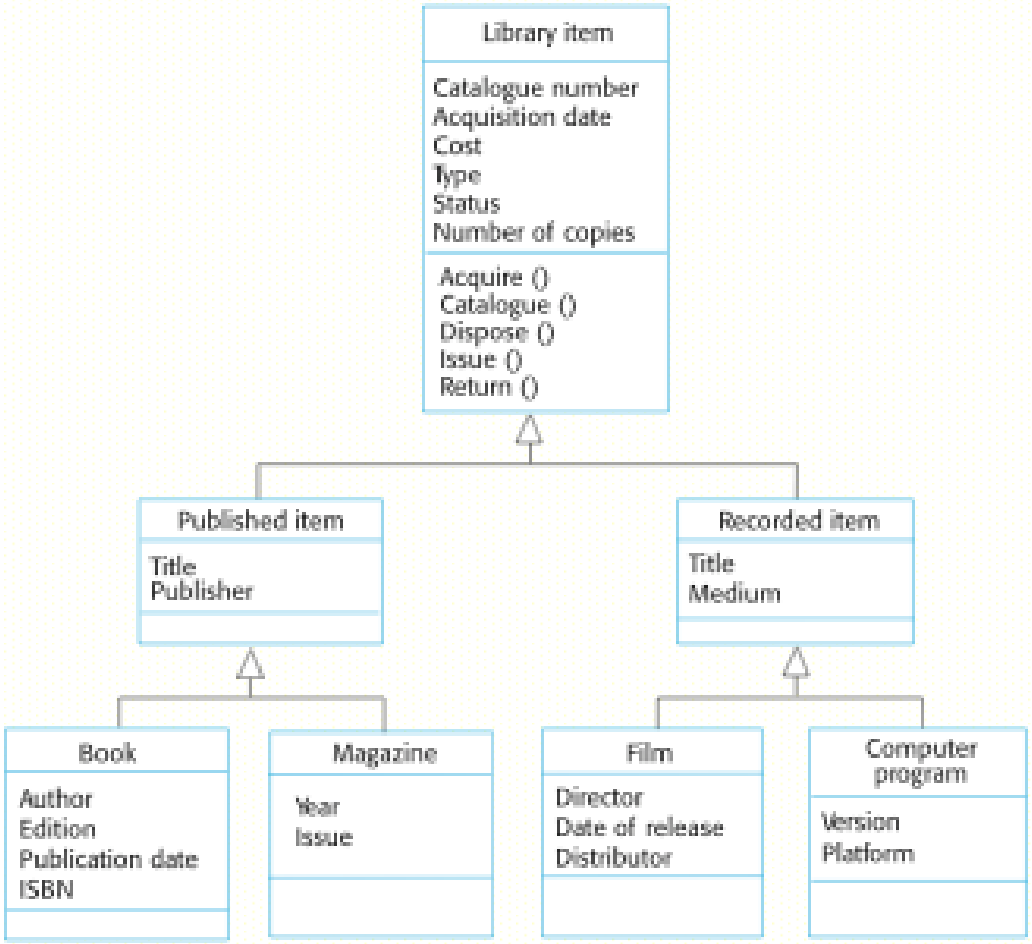
Object models

- Object models describe the system in terms of object classes and their associations.
- An object class is an abstraction over a set of objects with common attributes and the services (operations) provided by each object.
- Natural ways of reflecting the real-world entities manipulated by the system
- Object class identification is recognised as a difficult process requiring a deep understanding of the application domain
- Object classes reflecting domain entities are reusable across systems

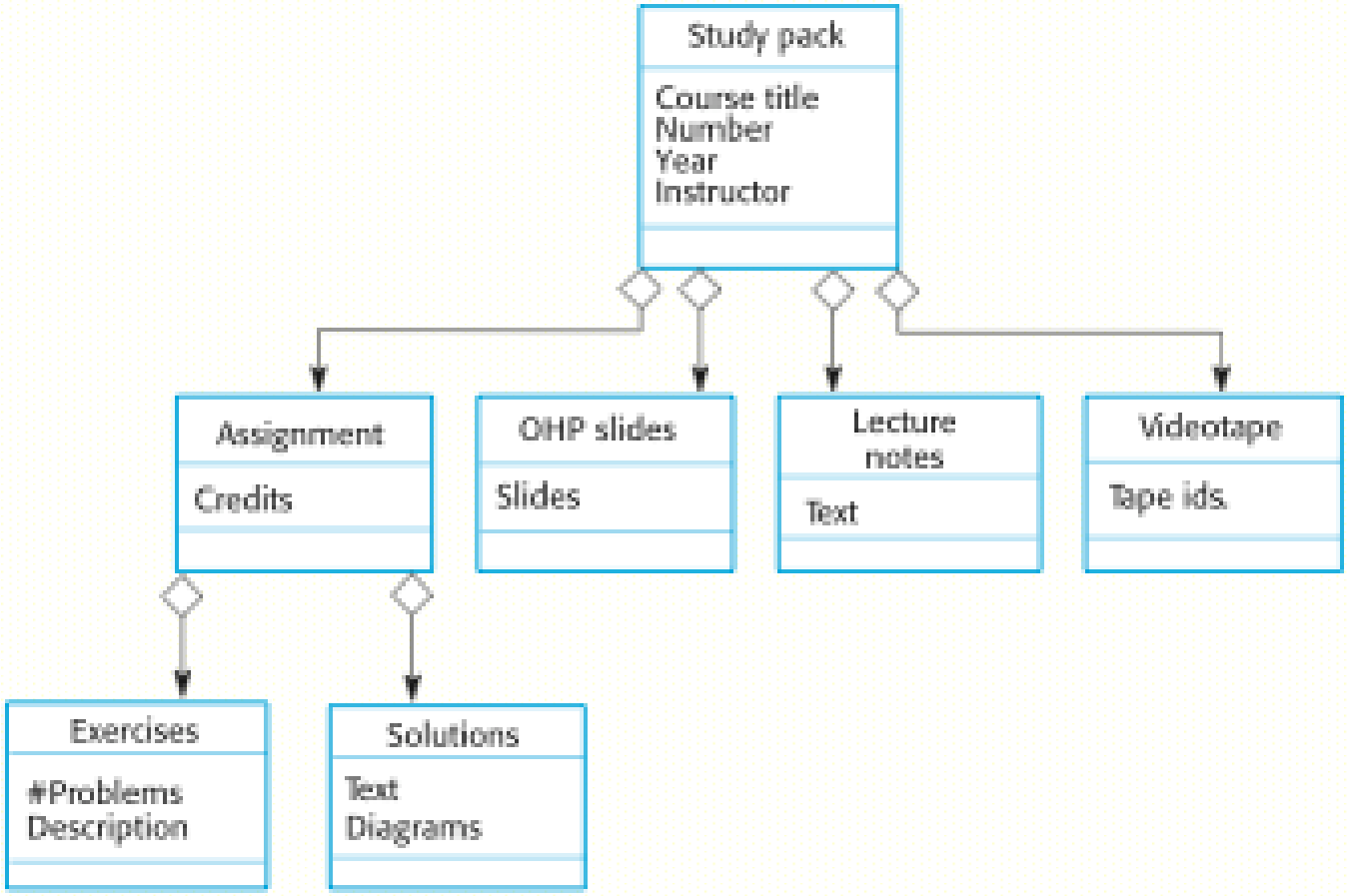
Object models and the UML

- Object classes are rectangles with the name at the top, attributes in the middle section and operations in the bottom section;
- Relationships between object classes (known as associations) are shown as lines linking objects;
- Inheritance is referred to as generalisation and is shown ‘upwards’ rather than ‘downwards’ in a hierarchy.

E.g., library class hierarchy



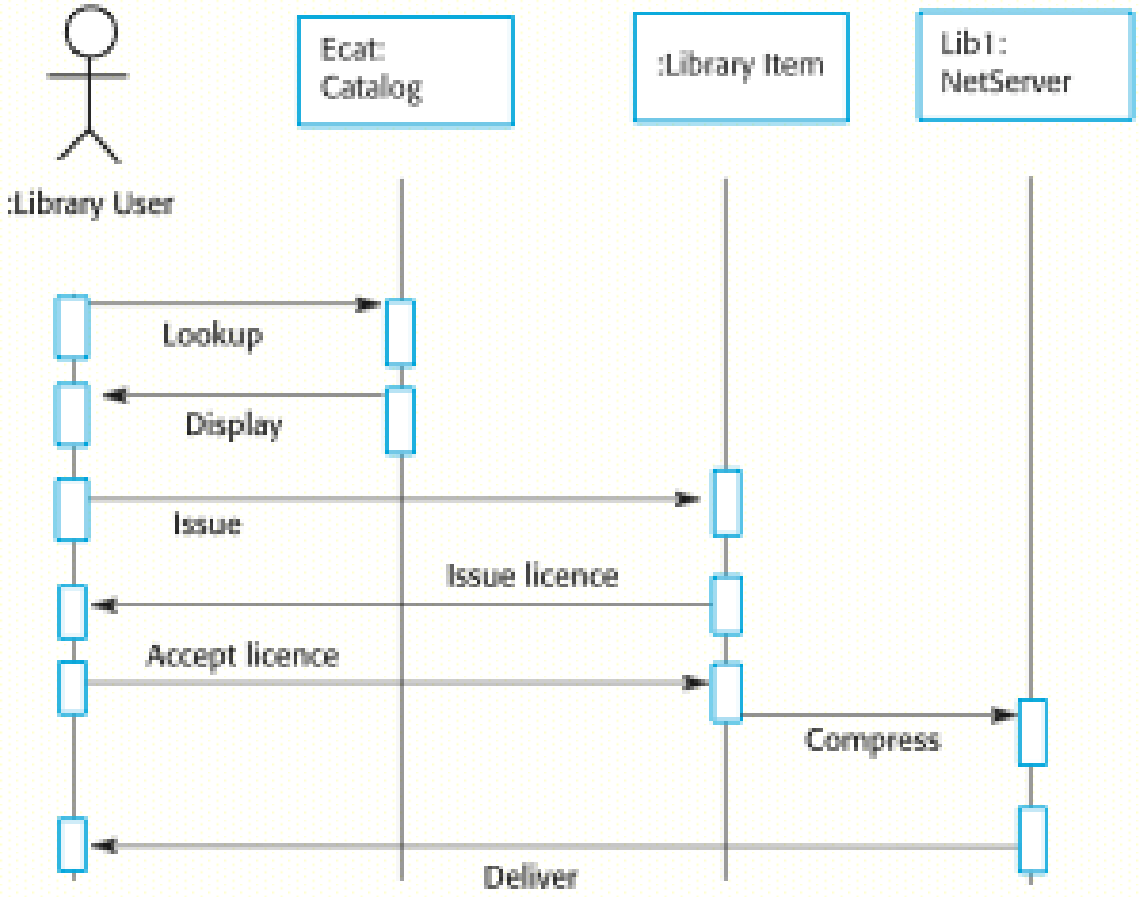
E.g., object aggregation



Object behaviour modelling

- A behavioural model shows the interactions between objects to produce some particular system behaviour that is specified as a use-case.
- Sequence diagrams (or collaboration diagrams) in the UML are used to model interaction between objects.

Issue of electronic items



Design Principles

Architecture & system characteristics

System non-functional characteristics may affect design. Examples:

- Performance
  - Localize critical operations and minimize communications. Use large rather than fine-grain components.
- Security
  - Use a layered architecture with critical assets in the inner layers.
- Safety
  - Localize safety-critical features in a small number of sub-systems.
- Availability
  - Include redundant components and mechanisms for fault tolerance.
- Maintainability
  - Use fine-grain, replaceable components.

Design Concepts

- Conceptual integrity / coherence
- Coupling / cohesion
- Information hiding
- Abstraction / refinement
- Rationale / tradeoffs

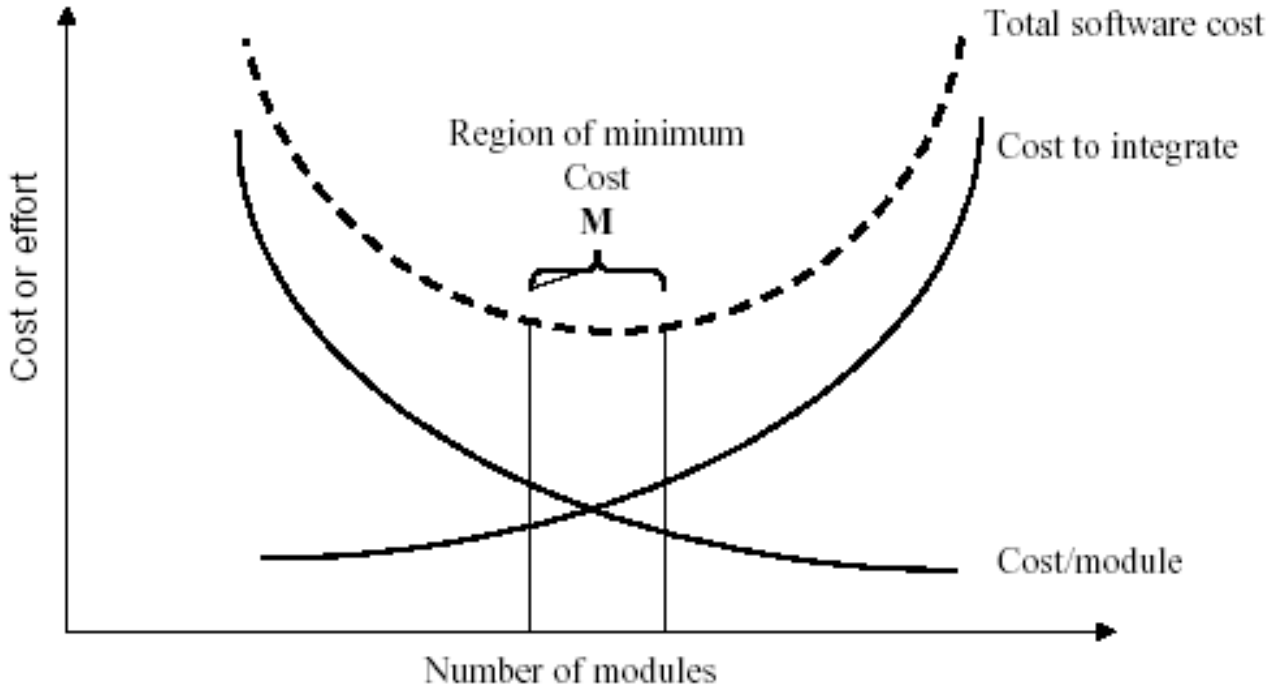
Coupling and cohesion

- *Coupling* - the extent to which two components depend on each other for successful execution
  - Low coupling is good
- *Cohesion* - the extent to which a component has a single purpose or function
  - High cohesion is good

Coupling and cohesion (cont'd)

- What does this mean:
  - modules single-minded/self-contained functions
  - address subset of requirements related to that function
  - simple interface, limited interaction ➡ bugs are often at interfaces
    - in terms of data, control, access to common content/data
  - easy to efficiently divide among team members

Modularity and Software Cost



- consider low coupling/high cohesion
  - module should be ‘stand alone’, errors contained as much as possible
- consider requirements
  - change in requirements should minimize number of modules affected

Information hiding

- Use of encapsulation to hide implementation details
- Reduce inter-component coupling thereby supporting subsequent maintenance

https://www.geeksforgeeks.org/encapsulation-in-java/

Abstraction and refinement

- All design methods support the idea of abstraction and refinement
- That is, designs are expressed at various levels of detail with correspondences between levels
- Various conceptual devices (abstraction mechanisms) are used to refine a design at one level to a lower level
- Abstractions include procedural, data and control abstraction

Rationale and tradeoffs

- Design decisions are explicit choices of how to trade-off two non-functional aspects of a design (e.g., speed versus size)
- Design decisions should be explicitly documented
- Documentation of design decisions is called *design rationale*

Design dimensions

- make architecture decision
  - repository, service, layered, …
- make decomposition decision
  - identify components
- determine control model
  - centralized, event-driven, …
- Describe modules/subsystems
  - Behavioral model
  - object model
  - …

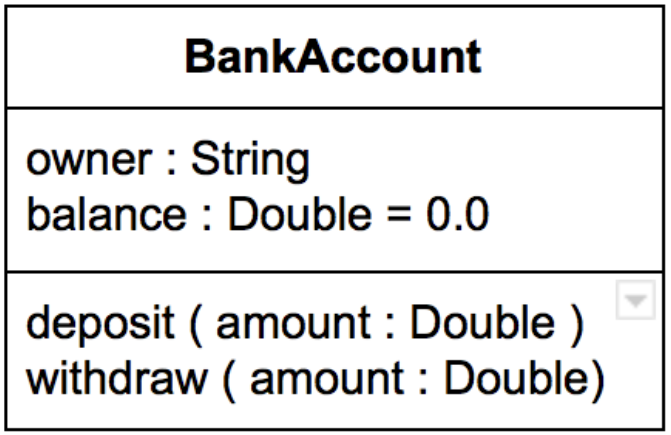
UML Class Diagrams
(+ exercise)

Modeling with UML Class Diagrams

- UML class model diagrams are commonly used to represent the structural aspects of system design problems
- A class diagram consists of a collection of object classes and the relationships among them

Classes

- A class is a distinct object type participating in the system being built
- A common noun in English often indicates an object type (i.e., a class)
- A class is represented by a rectangular box, possibly partitioned into three parts horizontally
  - Class name
  - Attributes
  - Operations



https://medium.com/@smagid_allThings/uml-class-diagrams-tutorial-step-by-step-520fd83b300b

Class Features

- Classes have features (attributes + operations)
- An attribute is a property of a class
  - Attributes have types that correspond to primitive or composite data types available on the computer
- An operation (method) is a service provided by a class. It may take parameters and return a value

Relationships

- Relationships exist among classes
- They are represented by lines connecting the related classes
- A transitive verb in English may indicate a relationship
- UML has three kinds of relationships
  - Generalization (is-a, class/subclass) 
  - Dependency (use) ------>
  - Association (consists-of)

Library Information System

- This exercise asks you to create a UML class diagram that models the problem of managing the information resources for a library
- Assume that somebody else will be designing the program from your analysis
  - Include classes, their attributes and operations and the relationships among them
  - Indicate attribute types, cardinality of associations, generalization and aggregation relationships

Library Problem Requirements

1. Each patron has one unique library card for as long as they are in the system.
2. The library needs to know at least the name, address, phone number, and library card number for each patron.
3. In addition, at any particular point in time, the library may need to know or to calculate the items a patron has checked out, when they are due, and any outstanding overdue fines.
4. Children (age 12 and under) have a special restriction–they can only check out five items at a time.
5. A patron can check out books or audio/video materials.
6. Books are checked out for three weeks, unless they are current best sellers, in which case the limit is two weeks.
7. A/V materials may be checked out for two weeks.
8. The overdue fine is ten cents per item per day, but cannot be higher than the value of the overdue item.
9. The library also has reference books and magazines, which can't be checked out
10. A patron can request a book or A/V item that is not currently in.
11. A patron can renew an item once (and only once), unless there is an outstanding request for the item, in which case the patron must return it.