Overview

- Introduction
- Design principles
- **Design methods**
- Conclusion

Design methods

- These design methods generally consist of a set of guidelines and procedures on how to go about designing a system.

Sample of design methods

Decision tables	Matrix representation of complex decision logic at the detailed design level.
E--R	Entity--Relationship Model. Family of graphical techniques for expressing data-relationships; see also chapter 10.
Flowcharts	Simple diagram technique to show control flow at the detailed design level. Exists in many flavors; see (Tripp, 1988) for an overview.
FSM	Finite State Machine. A way to describe a system as a set of states and possible transitions between those states; the resulting diagrams are called state transition diagrams; see also chapter 10.
JSD	Jackson System Development; see section 12.2.3. Successor to, and more elaborate than, JSP; has an object-oriented flavor.
JSP	Jackson Structured Programming. Data-structure-oriented method; see section 12.2.3.
LCP	Logical Construction of Programs, also known as the Warnier--Orr method; data-structure-oriented, similar to JSP.
NoteCards	Example hypertext system. Hypertext systems make it possible to create and navigate through a complex organization of unstructured pieces of text (Conklin, 1987).
OBJ	Algebraic specification method; highly mathematical (Goguen, 1986).
OOD	Object-oriented design; exists in many flavors; see section 12.3.
PDL	Program Design Language; example of a constrained natural language ('structured English') to describe designs at various levels of abstraction. Offers the control constructs generally found in programming languages. See (Pintelas and Kallistros, 1989) for an overview.

Sample of design methods

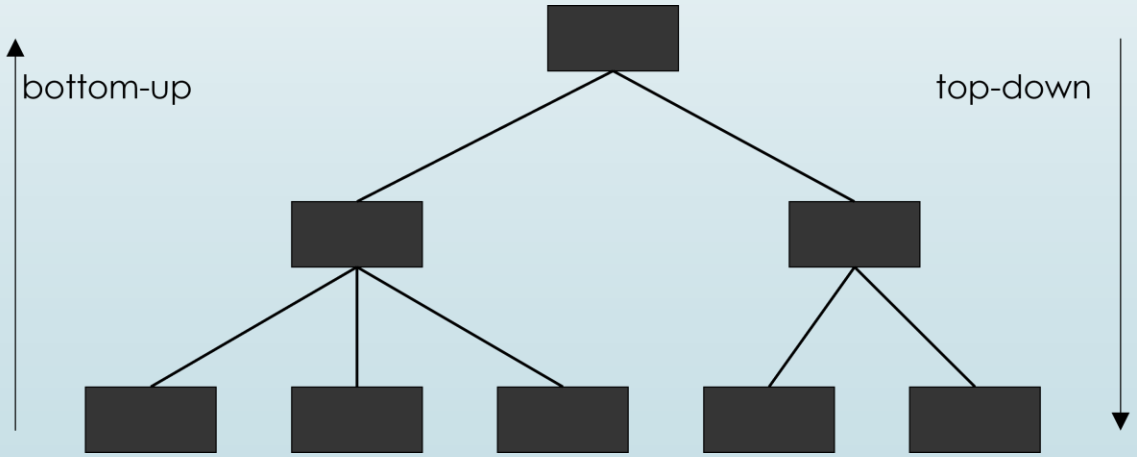
Petri nets	Graphical design representation, well-suited for concurrent systems. A system is described as a set of states and possible transitions between those states. States are associated with tokens and transitions are described by firing rules. In this way, concurrent activities can be synchronized (Peterson, 1981).
SA/SD	Structured Analysis/Structured Design. Data flow design technique; see also section 12.2.2.
SA/RT	Extension to Structured Analysis so that real-time aspects can be described (Hatley and Pirbhai, 1988).
SSADM	Structured Systems Analysis and Design Method. A highly prescriptive method for performing the analysis and design stages; UK standard (Downs et al., 1992).

Classical Design methods

- Functional decomposition

Functional decomposition

- In a functional decomposition the intended function is decomposed into a number of subfunctions that each solve part of the problem. These subfunctions themselves may be further decomposed into yet more primitive functions, and so on.
- Functional decomposition is a design philosophy rather than a design method



Functional decomposition (cnt'd)

- Extremes: bottom-up and top-down

Functional decomposition (cnt'd)

- Extremes: bottom-up and top-down

Top-down design Starting from the main user functions at the top, we work down decomposing functions into subfunctions.

Bottom-up design Using bottom-up design, we start from a set of available base functions. From there we proceed towards the requirements specification through abstraction.

Functional decomposition (cnt'd)

- Extremes: bottom-up and top-down

In its pure form, neither the top-down nor the bottom-up technique is likely to be used all that often. Both techniques are feasible only if the design process is a pure and rational one. And this is an idealization of reality

Functional decomposition (cnt'd)

- Extremes: bottom-up and top-down

In its pure form, neither the top-down nor the bottom-up technique is likely to be used all that often. Both techniques are feasible only if the design process is a pure and rational one. And this is an idealization of reality

- clients do not know what they want
- changes influence earlier decisions
- people make errors
- projects do not start from scratch, existing software might be used.

Classical Design methods

- Functional decomposition
- Data Flow Design (SA/SD)

Data flow design

Yourdon and Constantine (originated in early 70s)

- In its simplest form, data flow design is nothing but a functional decomposition with respect to the flow of data.
- component (module) is a black box which transforms some input stream into some output stream.
- In data flow design, heavily uses graphical representations known as Data Flow Diagrams (DFD) and Structure Charts.

Data flow design

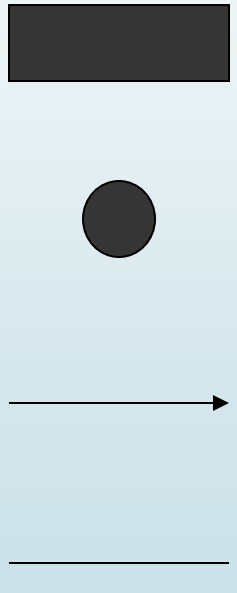
Structured Analysis and Structured Design (SA/SD) is a top-down decomposition technique system design methodology.

- two-step process:
  - First, a **logical design** is derived in the form of a set of **data flow diagrams**. During which a functional decomposition takes place. This step is referred to as **Structured Analysis (SA)**.
  - Next, the **logical design is transformed into a program structure** represented as a set of structure charts. During which module structure is formalized. The latter step is called **Structured Design (SD).**

Entities in a data flow diagram

The main result of Structured Analysis is a series of data flow diagrams. Four types of data entity are distinguished in these diagrams:

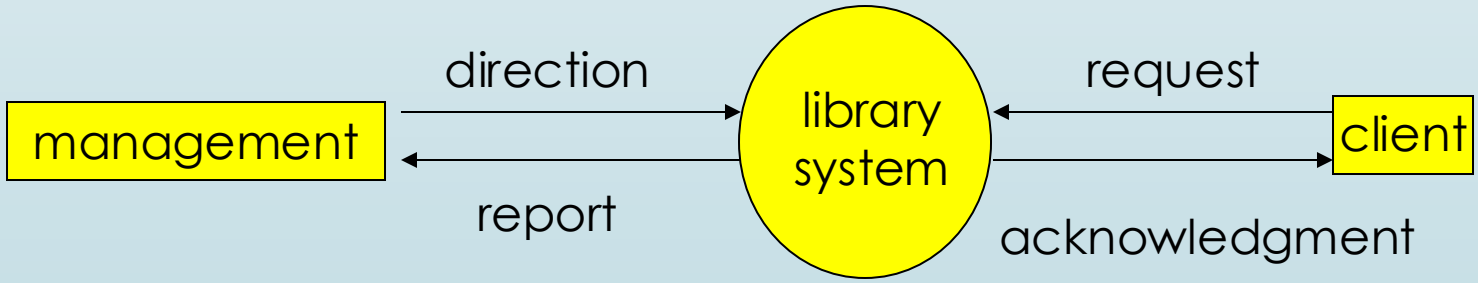
- External entities are the source or destination of a transaction. These entities are located outside the domain considered in the data flow diagram.
- Processes transform data. Processes are denoted by circles.
- Data flows between processes, external entities and data stores. Data flows are paths along which data structures travel.
- Data stores lie between two processes. This is indicated by the name of the data store between two parallel lines. Data stores are places where data structures are stored until needed.



Top-level DFD: context diagram

Example Simple library automation system which allows library clients to borrow and return books. It also reports to library management about how the library is used by its clients.

At the highest level we draw a context diagram. **A context diagram is a data flow diagram with one process,** denoting 'the system'. Its main purpose is to depict the interaction of the system with the environment (the collection of external entities).

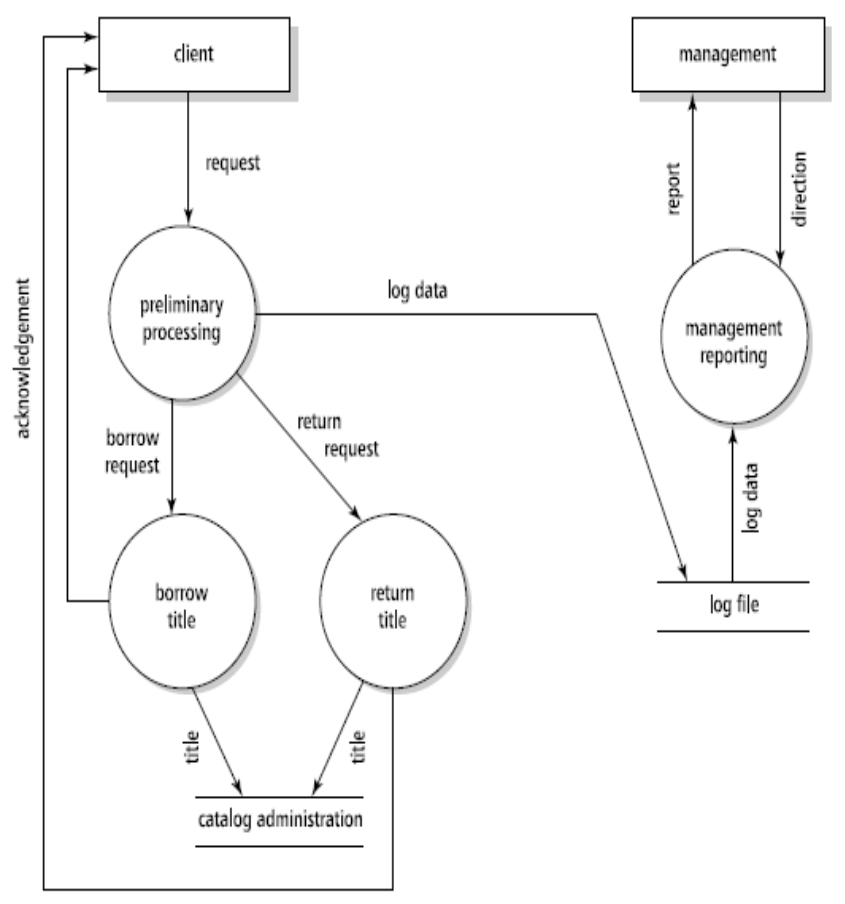


First-level decomposition

Next, this top-level diagram is further decomposed. In this diagram, we have expanded the central process node of the context diagram.

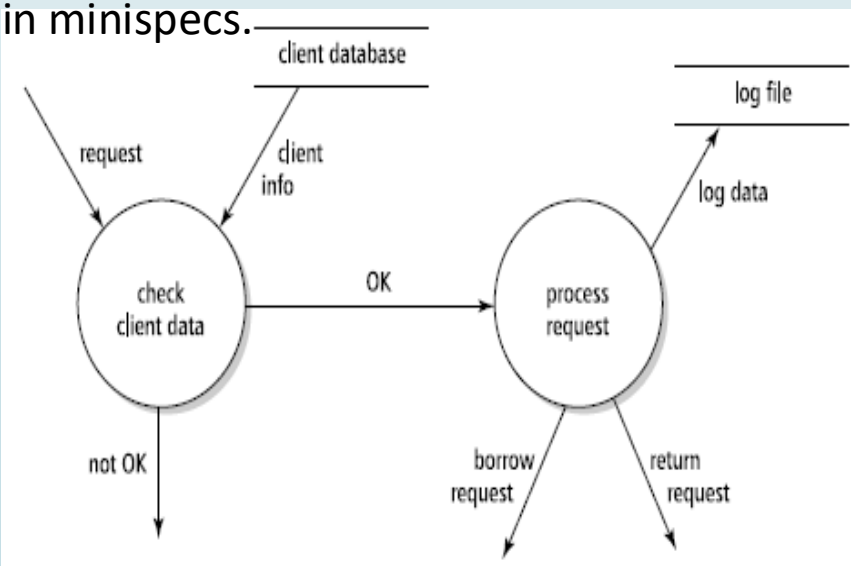
A client request is first analyzed in a process labeled 'preliminary processing'. As a result, one of 'borrow title' or 'return title' is activated. Both these processes update a data store labeled 'catalog administration'.

Client requests are logged in a data store 'log file'. This data store is used to produce management reports.



Second-level decomposition for "preliminary processing"

- These subsystems in turn are further decomposed into diagrams at yet lower levels.
- As an example, a possible refinement of the 'preliminary processing' node is given as below.
- In the lower level diagrams also, the external entities are usually omitted.
- These primitive processes are described in minispecs.



Minispec

- A Minispec serves to communicate the algorithm of the process to relevant parties. It may use notations like structured natural language, pseudocode, or decision tables.

Example minispec

```
Identification: Process request
Description:
1  Enter type of request
   1.1 If invalid, issue warning and repeat step 1
   1.2 If step 1 repeated 5 times, terminate transaction
2  Enter book identification
   2.1 If invalid, issue warning and repeat step 2
   2.2 If step 2 repeated 5 times, terminate transaction
3  Log client identification, request type and book
   identification
4  ...
```

Data dictionary entries

The contents of the data flows in a DFD are recorded in a data dictionary. It is nothing more than a precise description of the structure of the data. This is often done in the form of regular expressions.

```
borrow-request = client-id + book-id
return-request = client-id + book-id
log-data = client-id + [borrow | return] + book-id
book-id = author-name + title + (isbn) + [proc | series | other]
```

Conventions:

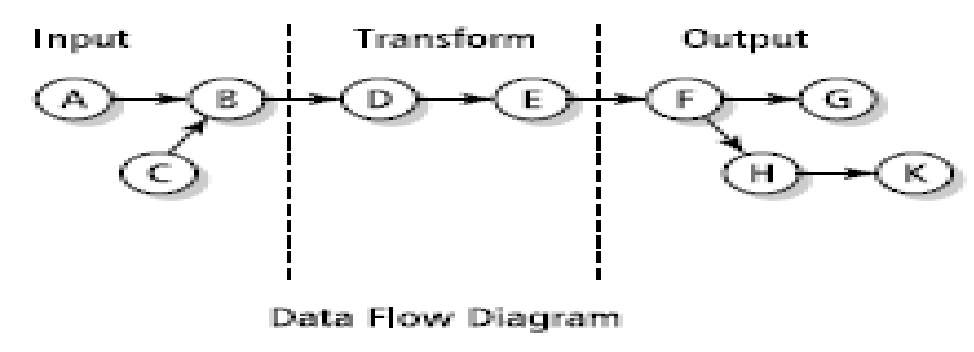
- [ ]: include one of the enclosed options
- | : separates options
- +: AND
- (): enclosed items are optional

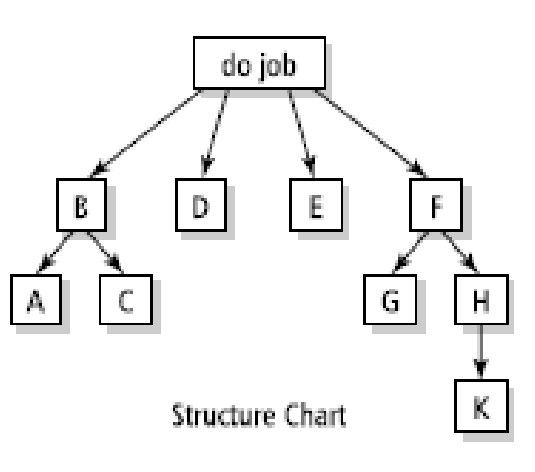
From data flow diagrams to structure charts

- The result of Structured Analysis is a logical model of the system. It consists of a set of DFDs, augmented by descriptions of its component in the form of minispecs, formats of data stores, and so on.

- Next, the data flow diagrams are transformed into a collection of modules (subprograms) that call one another and pass data. The result of the Structured Design step is expressed in a hierarchical set of structure charts.

Transform-centered design





Note that the arrows in a structure chart denote module calls, whereas the arrows in a data flow diagram denote flows of data.

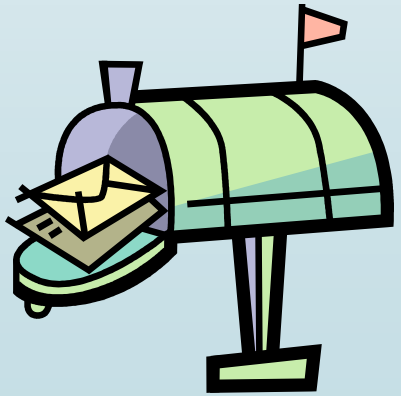
Classical Design methods

- Functional decomposition
- Data Flow Design (SA/SD)
- OO

OO Analysis and Design methods

three major steps:

1. identify the objects
2. determine their attributes and services
3. determine the relationships between objects

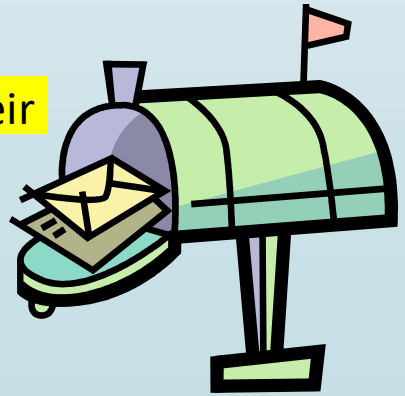


OO Analysis and Design methods

three major steps:

1. identify the objects
2. determine their attributes and services
3. determine the relationships between objects

The resulting picture of the system as a collection of objects and their interrelationships describes the **static** structure (decomposition) of the system.



(Part of) problem statement

Design the software to support the operation of a public library. The system has a number of stations for customer transactions. These stations are operated by library employees. When a book is borrowed, the identification card of the client is read. Next, the station's bar code reader reads the book's code. When a book is returned, the identification card isnot needed and only the book's code needs to be read.

(Part of) problem statement

Design the software to support the operation of a public library. The system has a number of stations for customer transactions. These stations are operated by library employees. When a book is borrowed, the identification card of the client is read. Next, the station's bar code reader reads the book's code. When a book is returned, the identification card isnot needed and only the book's code needs to be read.

Candidate objects

- software
- library
- system
- station
- customer
- transaction
- book
- library employee
- identification card
- client
- bar code reader
- book's code

Carefully consider candidate list

- eliminate implementation constructs, such as "software"
- replace or eliminate vague terms: "system" ⇒ "computer"
- equate synonymous terms: "customer" and "client" ⇒ "client"
- eliminate operation names, if possible (such as "transaction")
- eliminate individual objects (as opposed to classes). "book's code" ⇒ attribute of "book copy"

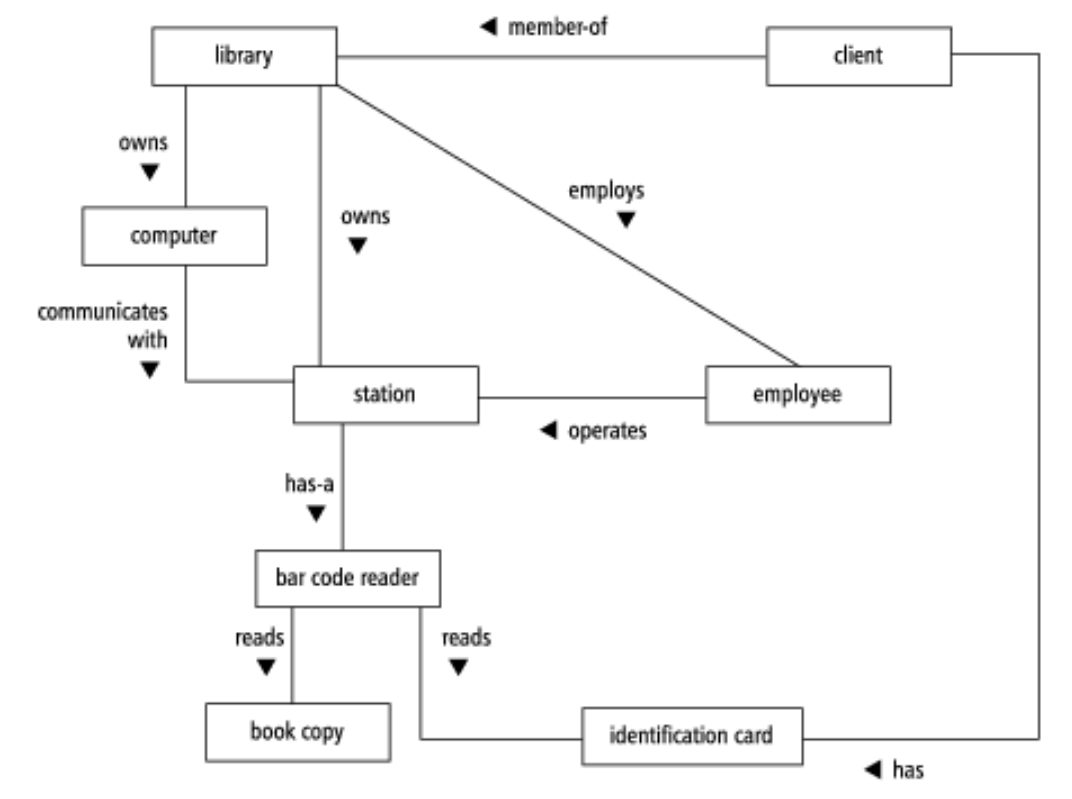
Relationships

- From the problem statement:
  - employee operates station
  - station has bar code reader
  - bar code reader reads book copy
  - bar code reader reads identification card
- Tacit knowledge:
  - library owns computer
  - library owns stations
  - computer communicates with station
  - library employs employee
  - client is member of library
  - client has identification card

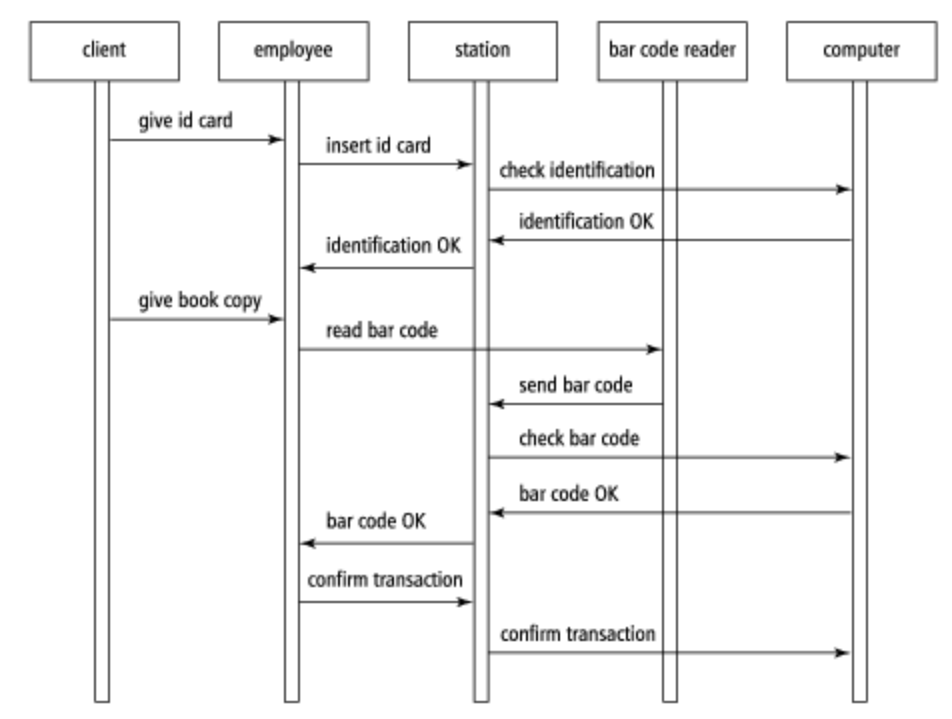
Draw a class diagram

- From the problem statement:
  - employee operates station
  - station has bar code reader
  - bar code reader reads book copy
  - bar code reader reads identification card
- Tacit knowledge:
  - library owns computer
  - library owns stations
  - computer communicates with station
  - library employs employee
  - client is member of library
  - client has identification card

Result: initial class diagram



Usage scenario ⇒ sequence diagram



when choosing a particular design method …

- Familiarity with the problem domain
- Designer's experience
- Available tools
- Development philosophy

Overview

- Introduction
- Design principles
- Design methods
- **Conclusion**

Conclusion

- Essence of the design process: decompose system into parts
- Desirable properties of a decomposition: coupling/cohesion, information hiding, (layers of) abstraction
- There have been many attempts to express these properties in numbers
- Design methods: functional decomposition, data flow design, data structure design, object-oriented design