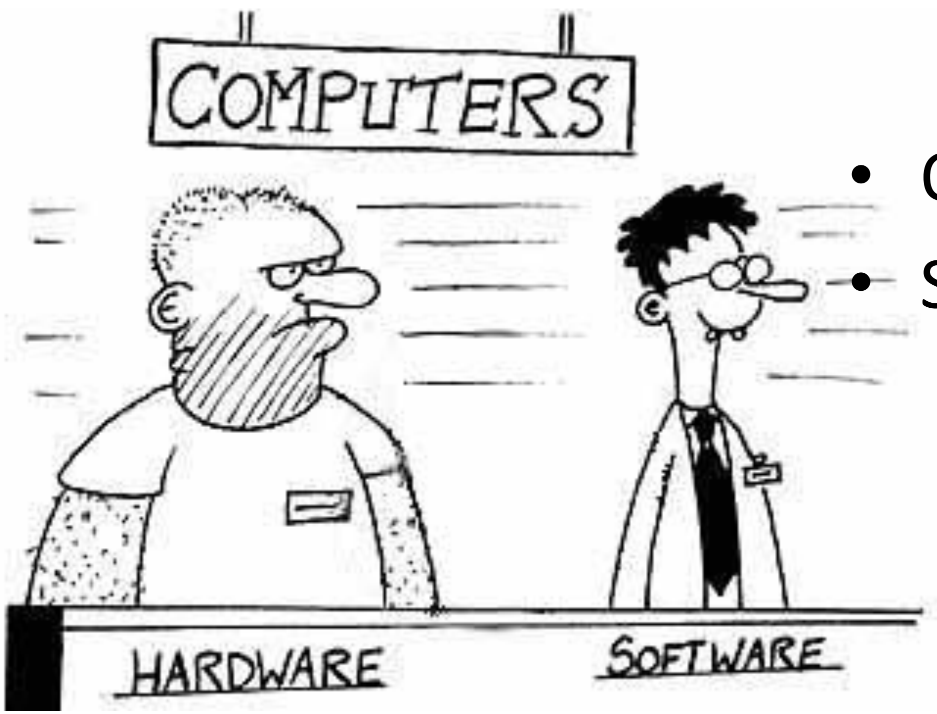
PART 2

Software



- Operating Systems
- Software Engineering
- Algorithms
- Programming Languages

- **They will be covered in labs or in advance courses**

- **Note:** ***hardware & software are logically equivalent!***

PART 2

Software

- **Hardware & software logically equivalent:**
  – Any operation performed by software can also be built directly into hardware
  – Any instruction executed by hardware can also be simulated in software

CHAPTER 6

Software Engineering

Reference: Computer Science an Overview
Author: J. Glenn Brook Shear
6th Edition

- Building LARGE / complex software systems

6.1: Engineering Example

- Design
- Re-design
- Construction
- Integration of parts
- Materials
- Transportation
- Financing
- Time assessment
- Personnel
- Politics
- Drawings/Documentation
- …

6.1: Software Engineering

- Building large software systems is engineering effort too, incl.
  - division of problem into manageable parts
  - integration of separately developed units
  - cost assessment (time / money, …)
  - personnel management…
- But it’s not exactly identical
- In traditional engineering pre-defined rules-> Off the shelf components

6.1: Software vs. Real-world Engineering

- SE differs from real-world engineering:
  - reuse of pre-fabricated parts often not possible
    - so, large systems often built from scratch
  - software is either correct or incorrect
    - no tolerances, as in real-world ‘objects’
  - ‘quality’ of software is hard to define / measure
    - real-world measure: how well does object endure strain over time?
  - software does not wear out…
    - … but it can become outdated

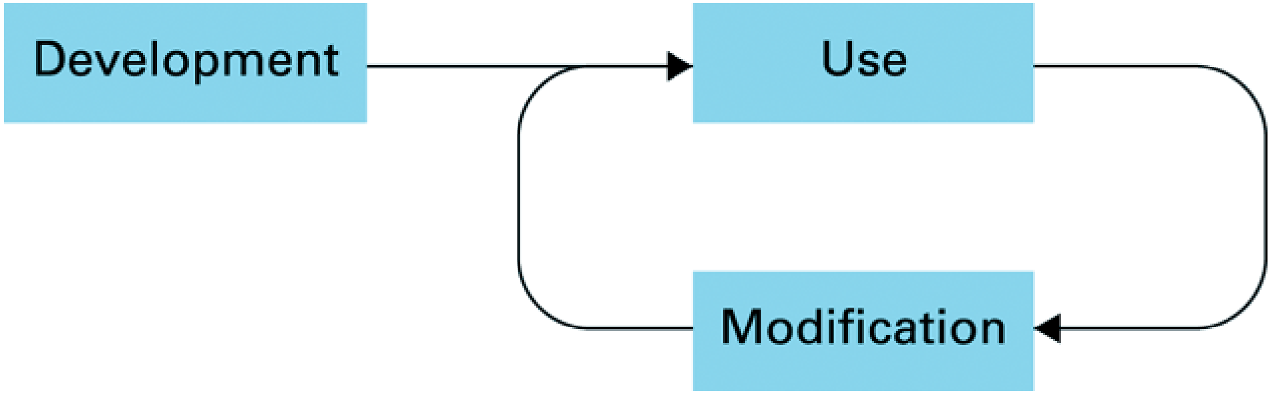
6.1: Large/complex software systems

- Common idea/mistake:
  - Large/complex = many lines of code
- More realistic:
  - Large/complex = many interrelated entities that need to work together as a single system
- Note:
  - goal of software engineering is to make such systems ‘manageable’

Research in Software Engineering

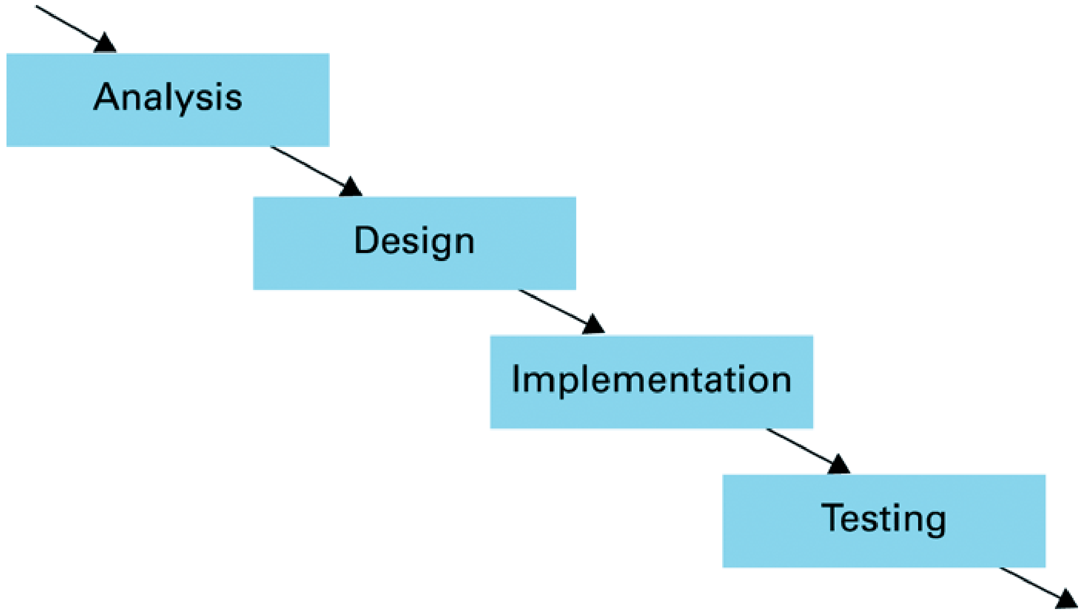
- **Two Levels**
  - **Practitioners:** Work toward developing techniques for immediate applications.
  - **Theoreticians:** search for underlying Principles and theories on which more stable techniques can someday be constructed.
- Both needed.
- **ACM & IEEE**.

6.2: The Software Life Cycle



- For real-world objects: modification = repair
- Modification phases combined often much more costly than development phase
  - ‘modification’ often is: ‘redesign from scratch’
  - note: comments in code are essential

6.2: Development Phase



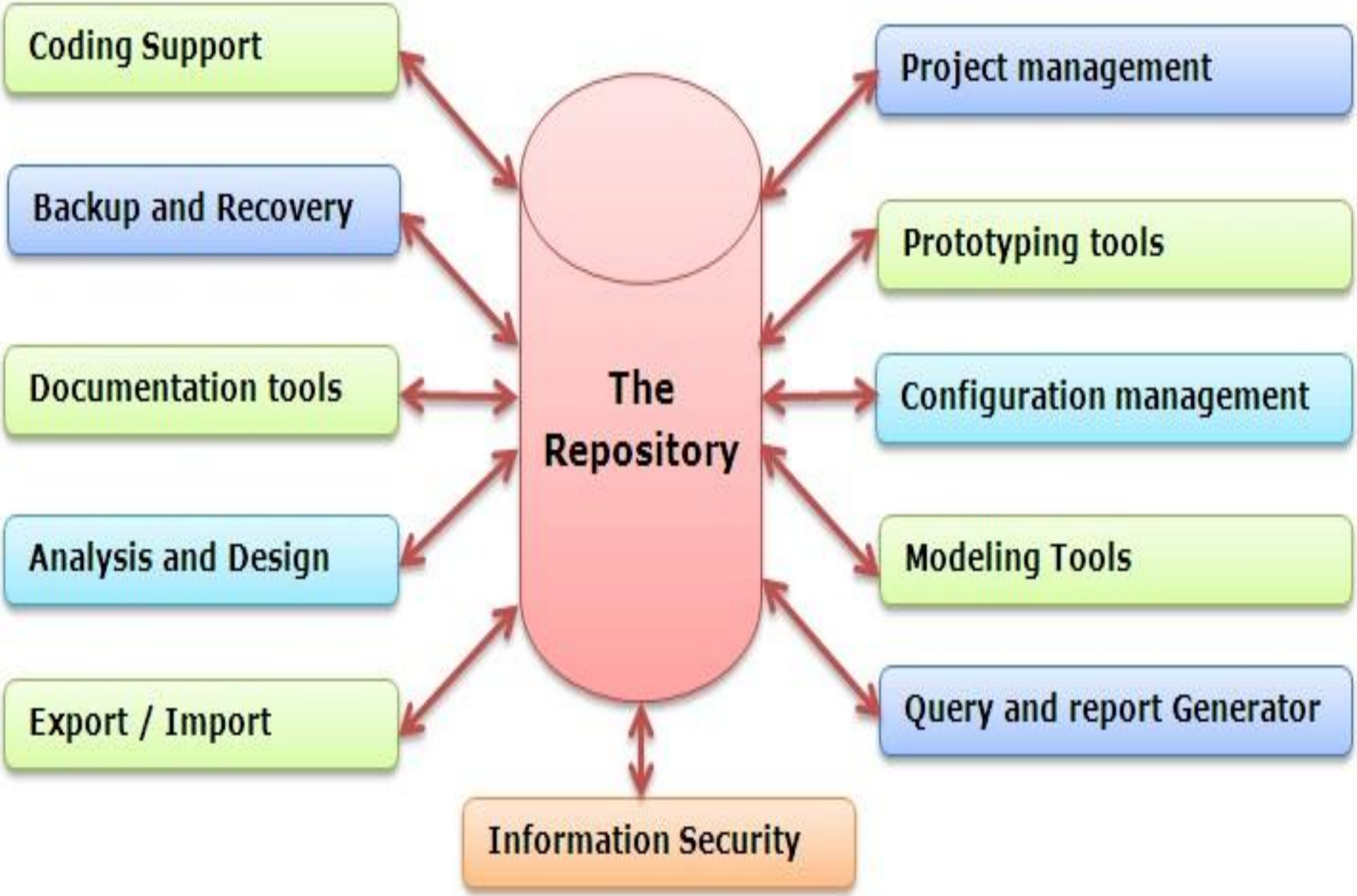
- Compare: ‘art of problem solving’
- General cost estimate (in time):

Analysis: 30% - Design: 20%

Implementation: 10% - Testing: 40%

Trends in Software Engineering

- Water Fall Model
- Incremental Model
- Prototyping
  - Evolutionary Prototyping
  - Throwaway Prototyping
    - Rapid Prototyping
- CASE (Computer-Aided Software Engineering)
  - Project Planning Tools

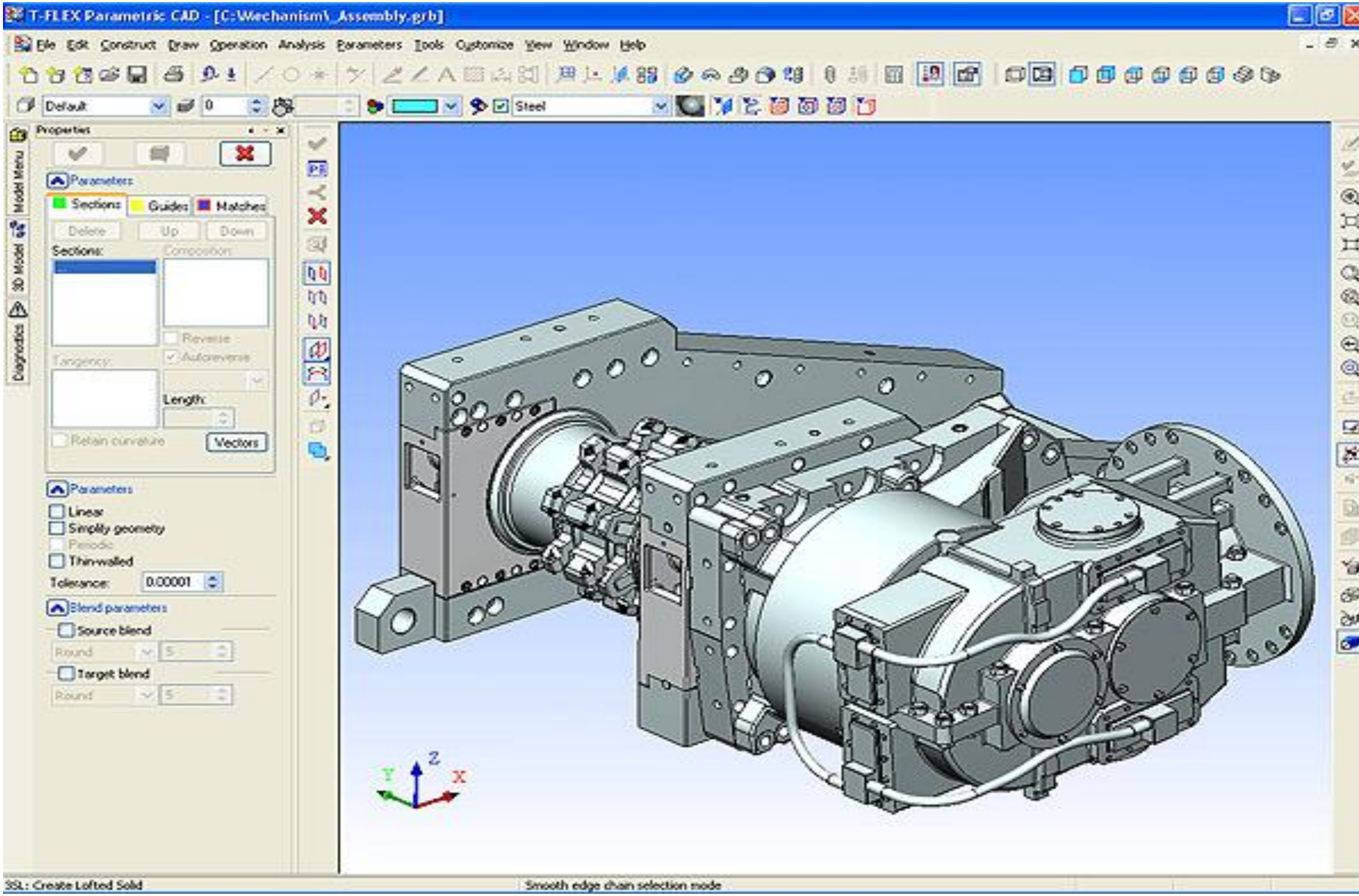

Coding Support
Backup and Recovery
Documentation tools
Analysis and Design
Export / Import
The Repository
Project management
Prototyping tools
Configuration management
Modeling Tools
Query and report Generator
Information Security

CASE Tools

Trends in Software Engineering (2)

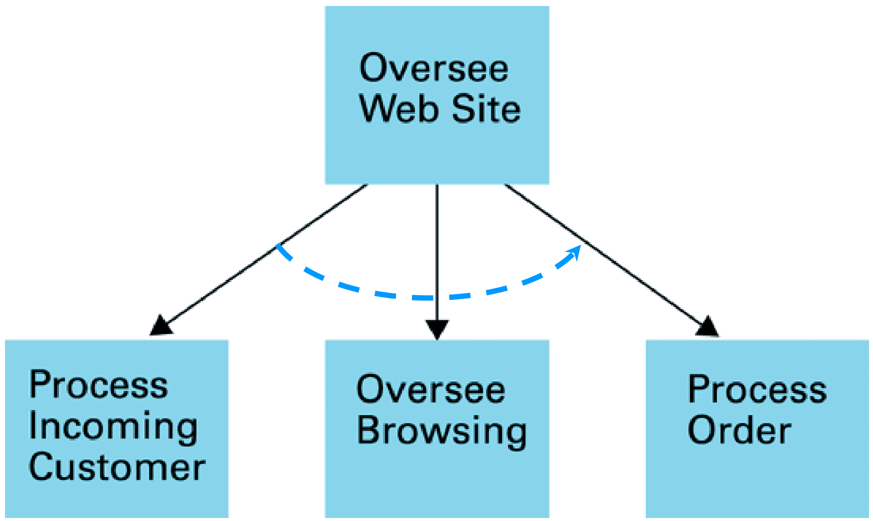
- – Project Management Tools
- – Documentations Tools
- – Prototyping & Simulation Tools
- – Interface Design Tools
- – Programming Tools

Computer Aided Design



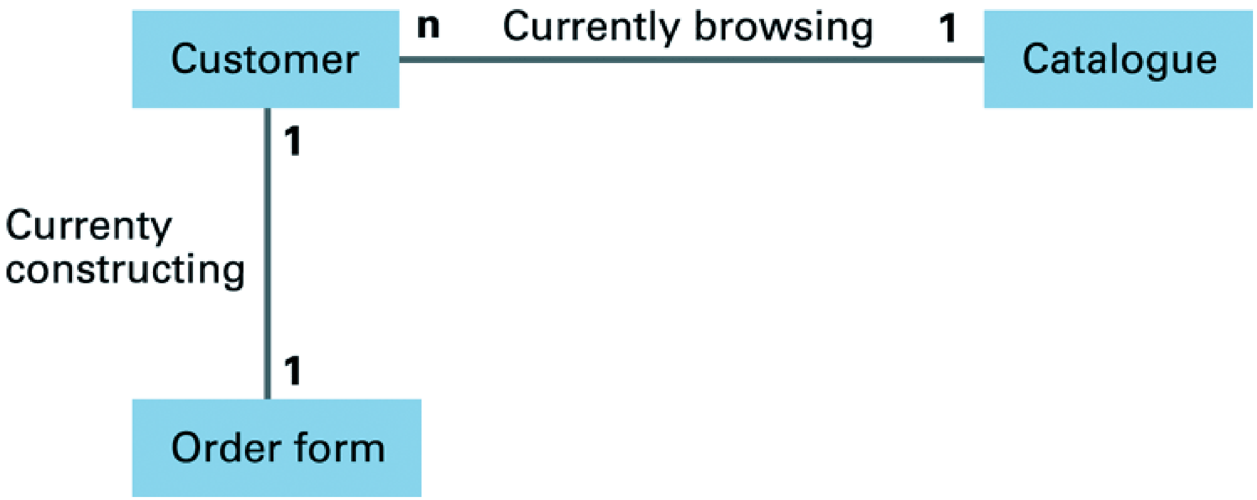
Modularity

- Modularity:
  - division of software into manageable parts, each of which performs a subtask only
    - e.g.: procedures, objects, …
- Representation of procedural modularity
  - structure chart:



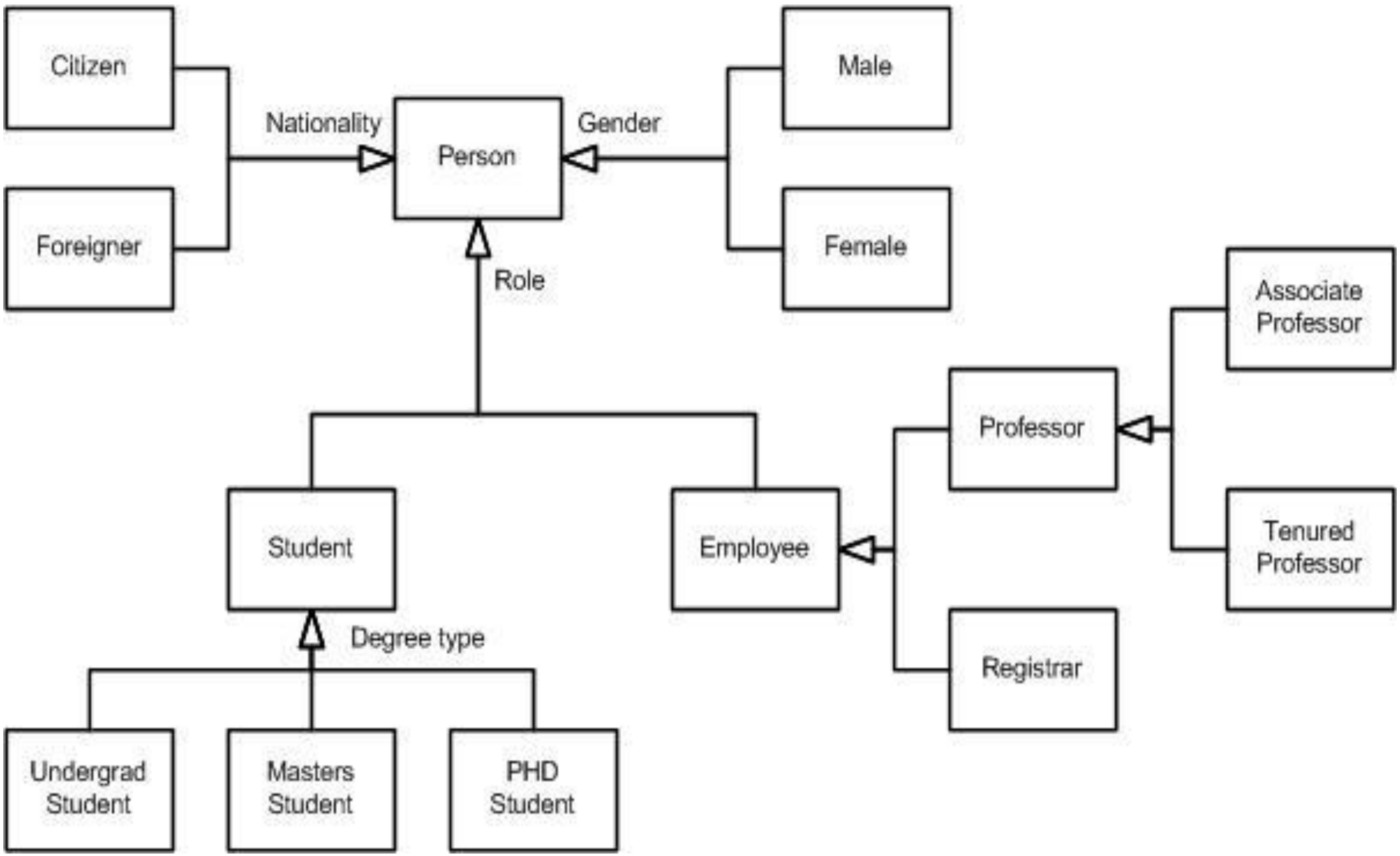
6.3: Modularity in OO systems

- Representation of object-oriented modularity
  - class diagram:



- Objects related by 'relationships' (i.e.: methods)
  - here: *one-to-one* and *one-to-many*
  - *UML* Conventions

UML Conventions



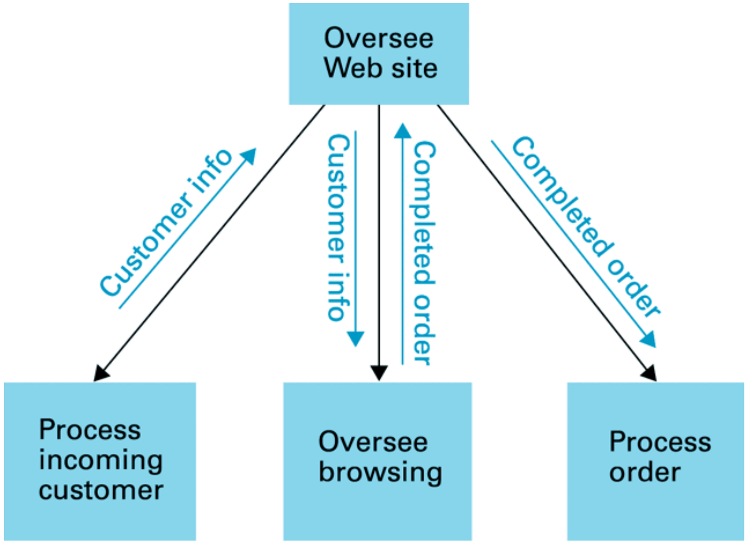
6.3: Inter-Module Dependencies (1)

- Modularity is to obtain *maintainable* software
  - future modifications will only affect few modules

- Note:
  - only when dependence between modules in minimized
- Unfortunately:
  - some dependency always needed to form a coherent system

6.3: Inter-Module Dependencies (2)

- Dependency between modules known as
  - 'coupling'
- Two forms:
  - 'control coupling'
    - passing of control from one module to another
    - i.e.: sequence of procedures called
  - 'data coupling'
    - sharing of data between modules
    - i.e.: data passed as parameters



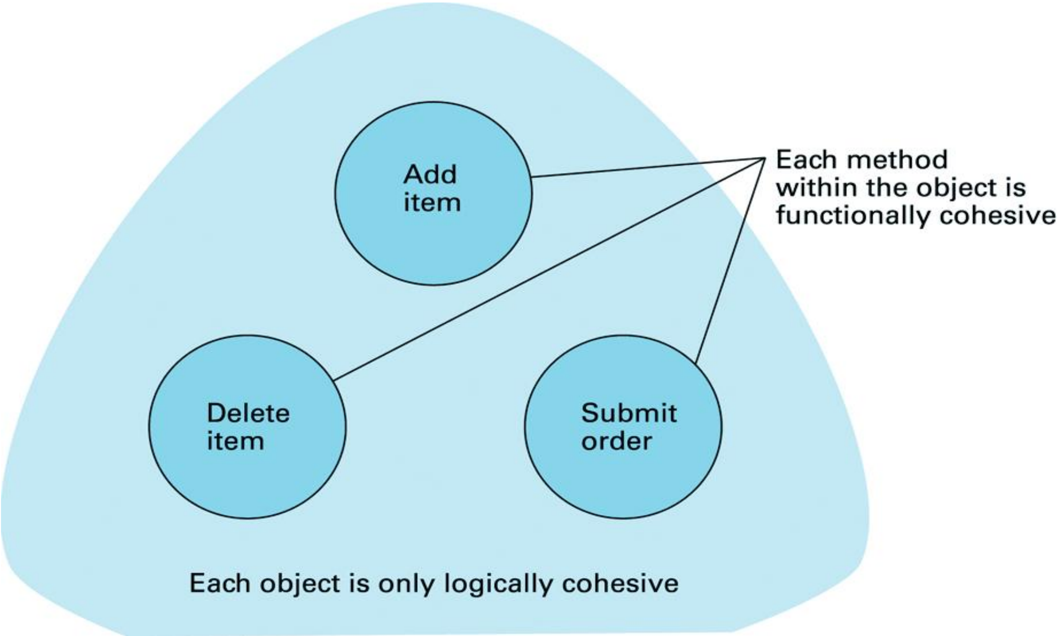
6.3: Inter-Module Dependencies (3)

- Note: main benefit of OO-design:
  - data coupling is minimized
  - inter-object dependencies by method invocations (i.e.: control coupling)
- Danger: ‘implicit coupling’
  - by ‘global’ data that are accessible by all modules
  - difficult to trace global data accesses and updates
- So: minimize use of global data!

6.3: Intra-Module Dependencies

- Also important:
  - maximize bindings (or ‘cohesion’) *within* a module
  - ‘put together what belongs together’

- In OO:



Cohesion forms:

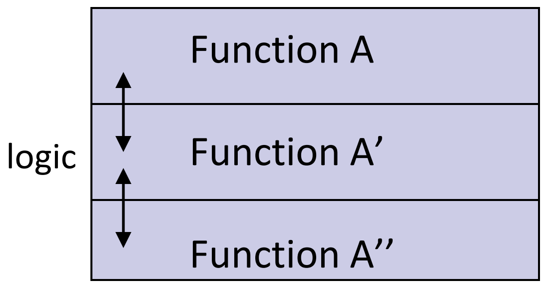
- **Logical Cohesion**: Within a module induced by the fact that its internal elements perform activities logically similar in nature.
- Elements of component are related logically and not functionally.

Example

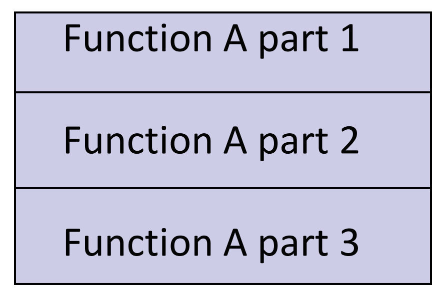
- A component reads inputs from tape, disk, and network.
- All the code for these functions are in the same component.
- Operations are related, but the functions are significantly different.

- **Functional Cohesion:** All the parts of a module are geared towards the performance of a single activity.
- Every essential element to a single computation is contained in the component.
- Every element in the component is essential to the computation.
- Ideal situation
- What is a functionally cohesive component?
  - One that not only performs the task for which it was designed but
  - it performs only that function and nothing else.

Logical Vs Functional Cohesion



Logical
Similar functions



Functional
Sequential with complete, related functions

Chapter 6 - Problem 13

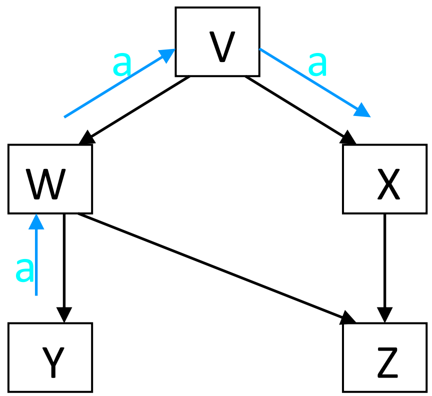
Which is an argument for *coupling*, and which for *cohesion*:

a) For a student to learn, a subject should be presented in well-organized units with specific goals.
b) A student does not really understand a subject until the subject's overall scope and relationship with other subjects has been grasped.

- Coupling => b
  - relationships among subjects are like data coupling
- Cohesion => a
  - well-organized internal subject structure similar to logical cohesion

Chapter 6 - Problem 16

Answer in relation to the following structure chart:



a) To which module does module Y return control? => W

b) To which module does module Z return control? => W, X

c) Are modules W and X linked via control coupling? => no

d) Are modules W and X linked via data coupling? => yes

e) What data is shared by both module W and module Y? => parameter ‘a’

f) In what way are modules Y and X related? => data coupling