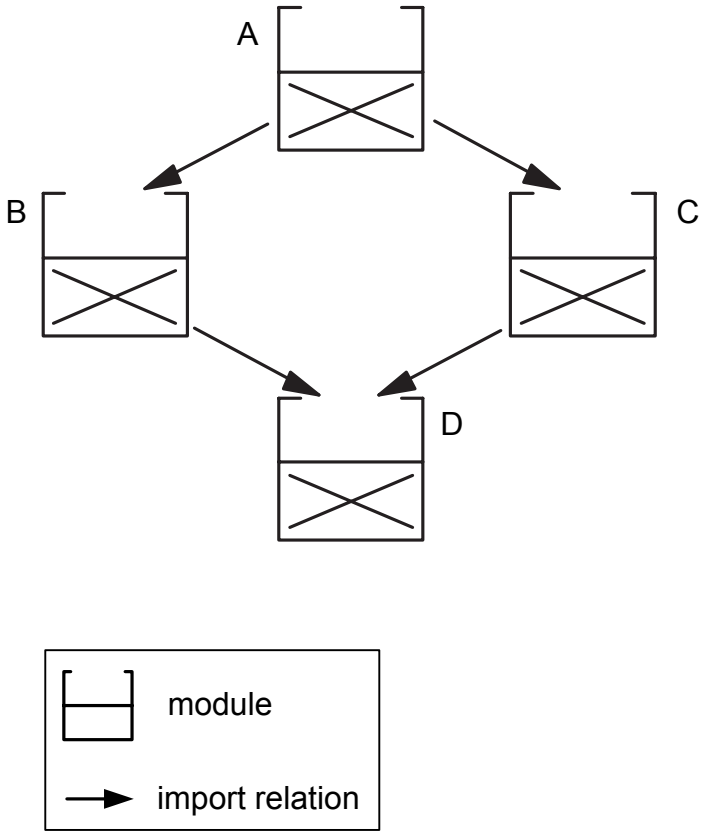
2. The Problem: Modelling on Design Level

Aims

- Explain importance of architecture modelling
- Motivation for architecture language
- Engineering-like architecture modelling

Contents of Chapter 2

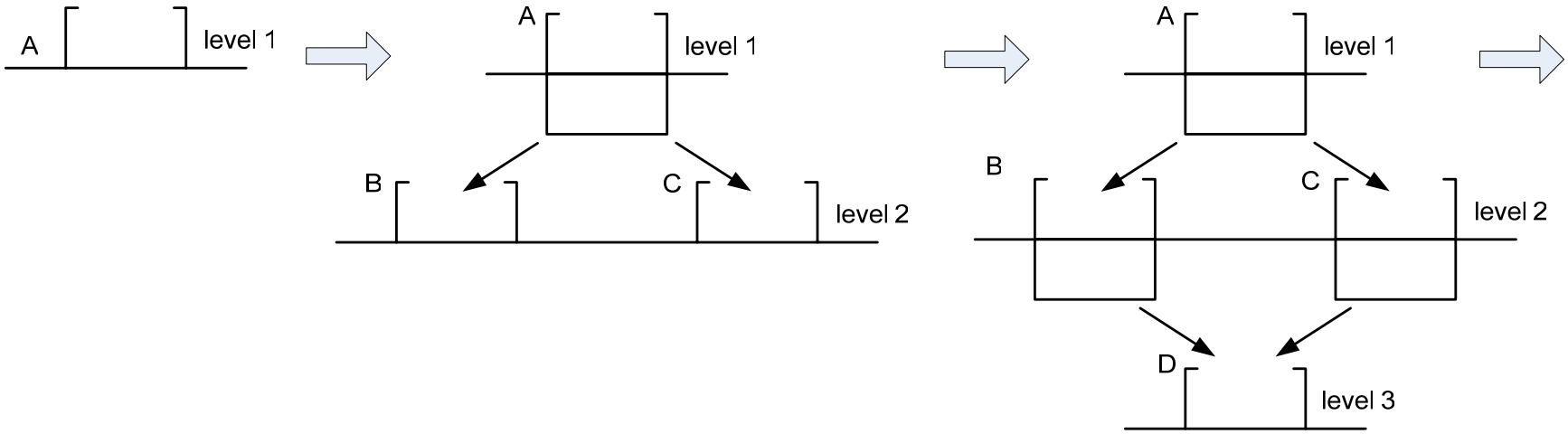
Architecture Paradigm - an Idealization



- Idealization:
  - » Design before implementation
  - » All components of all layers: only interfaces
- Problems:
  - » Identification of
    - ⇨ All layers
    - ⇨ All components

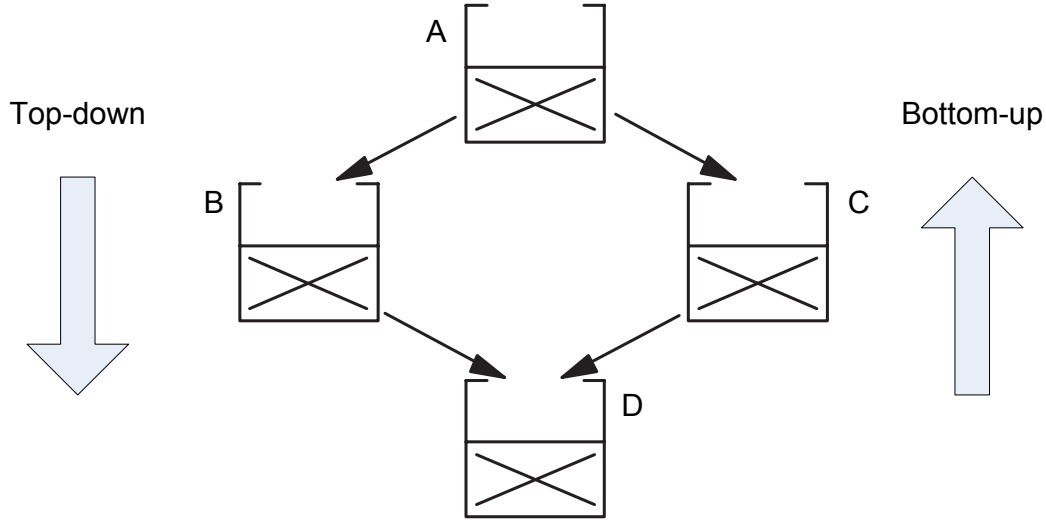
    } without realization
  - » Changes will happen:
    - ⇨ Components too complex
    - ⇨ trivial components

Alternative: Continuous Paradigm?



- Problems:
  - » No division of labor
  - » No architecture:
    - ⇨ No quality assurance before realization
    - ⇨ No reuse of components

Design Strategies



- Top-Down
- Bottom-Up
- Jo-Jo

Software Systems Are Wrong

N	p	$P = p^N$
10	0,99	0,9
10	0,9	0,35
100	0,99	0,37
100	0,9	0,000027

Correctness of a software system (Dijkstra)

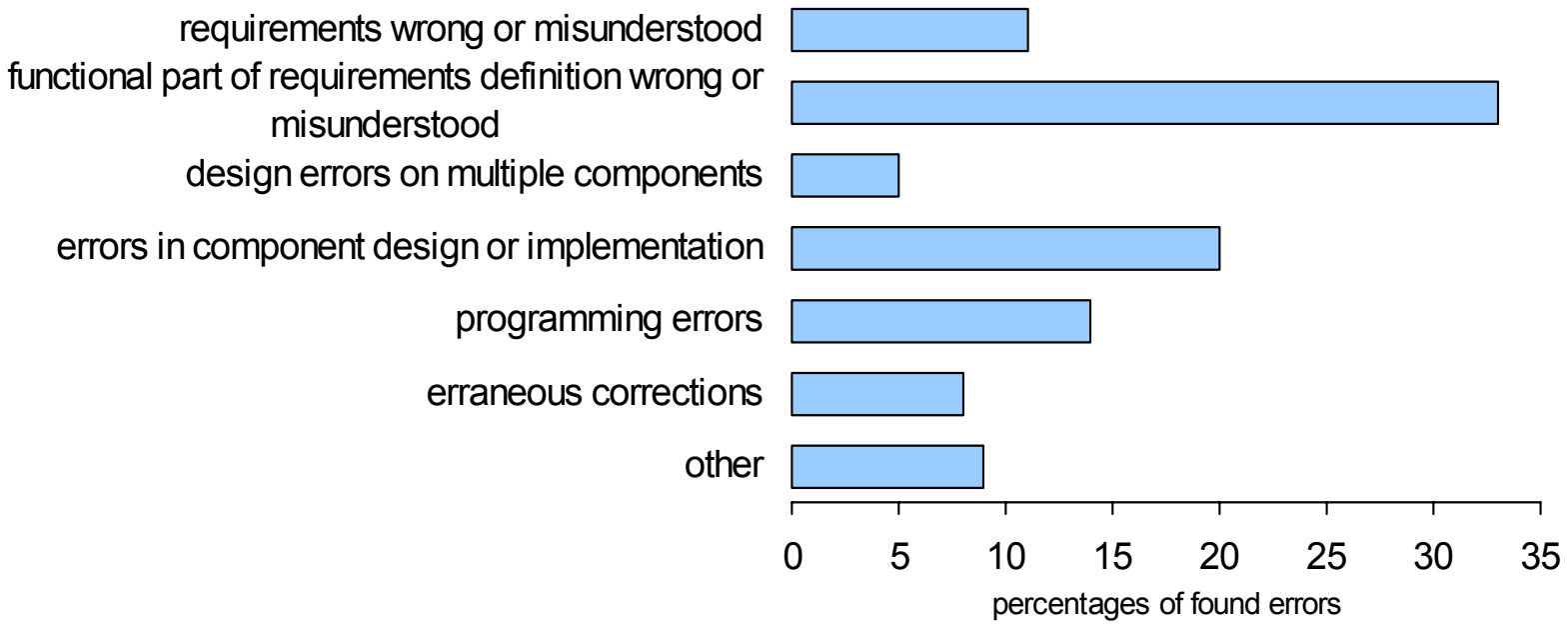
- Errors principal definition:
  - Wrong: statically wrong, principally wrong
  - Error: anywhere, anytime
- Errors practical definition:
  - Time from error to error: acceptable
  - error correction effort < benefit of system runs

Types of Errors

- Requirements:
  - » Requirements specification not reflecting needs
  - » Requirements specification incomplete, contradicting
- Architecture:
  - » Does not meet requirements
  - » Does not reflect SE quality measures: adaptability, portability, etc.
- Implementation:
  - » Component realization not consistent to design specification
  - » Realization wrong or inefficient
- Documentation:
  - » Analysis and design decisions not documented
  - » Documentation not consistent with technical system
- Project organization:
  - » Bad estimation, qualified developers unavailable, no clear responsibilities, no precise monitoring, no risk analysis

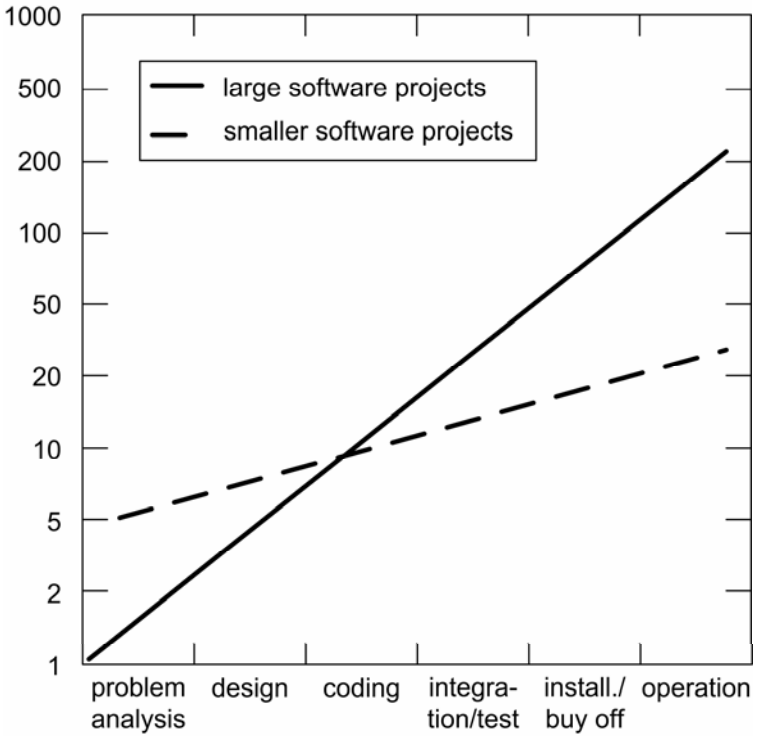
Design and Related Errors

- Minor effort for requirements engineering and design → many errors, high maintenance costs
- More than half of errors in early phases



Relative Costs of Errors

Relative costs



Errors detected in

- Relative and not absolute costs
- Error must have been made before detected

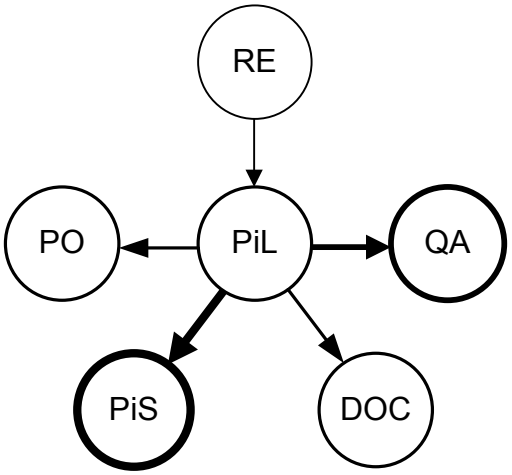
  ⇨ Avoid requirements and design errors

Conclusions

- More effort and competence
  - » More effort for early phases
  - » Suitable notations / methods necessary
  - » Experience and tools

  } *
- Better quality assurance
  - » Inspection of design results
  - » capability to evaluate demands for *
- Modifiable architectures
  - » Errors cause changes
  - » Changes are likely to happen
  - » Suitable architectures demand for *

Master Document “Architecture”



Thickness of circle corresponds to effort in development process

Thickness of edge: determination of results in dependent working area

⇨ development process mainly determined by design activity

- Architecture has to map requirements
- Architecture determines long-term quality of system
- One architecture can answer different requirements

- Organization of development
- Organization of change
- Design errors have influence

All Developers Have to Understand the Architecture

Purpose

- Graphical notation: for overview
- Textual notation: for design details, i.e. interface description

Both are a tremendous abstraction from source code

Architecture

- To estimate the effort of changes and to plan changes
- To carry out changes
- To study the structure of a system
- To identify reusable components, patterns, or standard solutions
- To define subprocesses in other working areas: implementation, quality assurance, integration, and documentation

Architecture and Project Organization

Tasks:

Programming in the large (Design) ⟺ Programming in the many (Project organization)

Control of versions, variants, configurations

Organization of versions / variants, configurations esp. important for architecture and units

---

Roles:

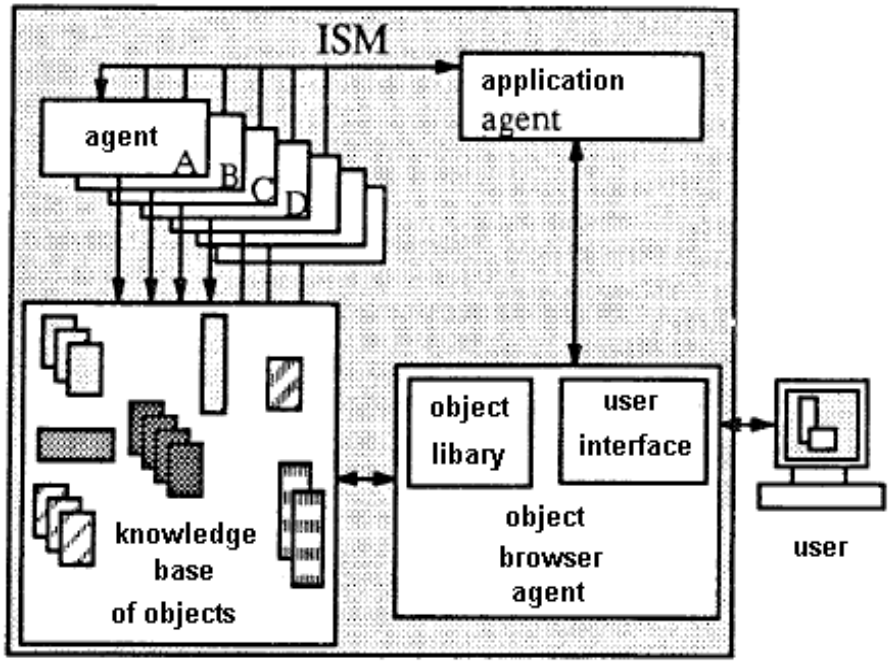
Technical side
chief programmer / architect

Organizational side
project manager

Maybe the same person

Architectures Are More Than Pictures

- What are architectures?

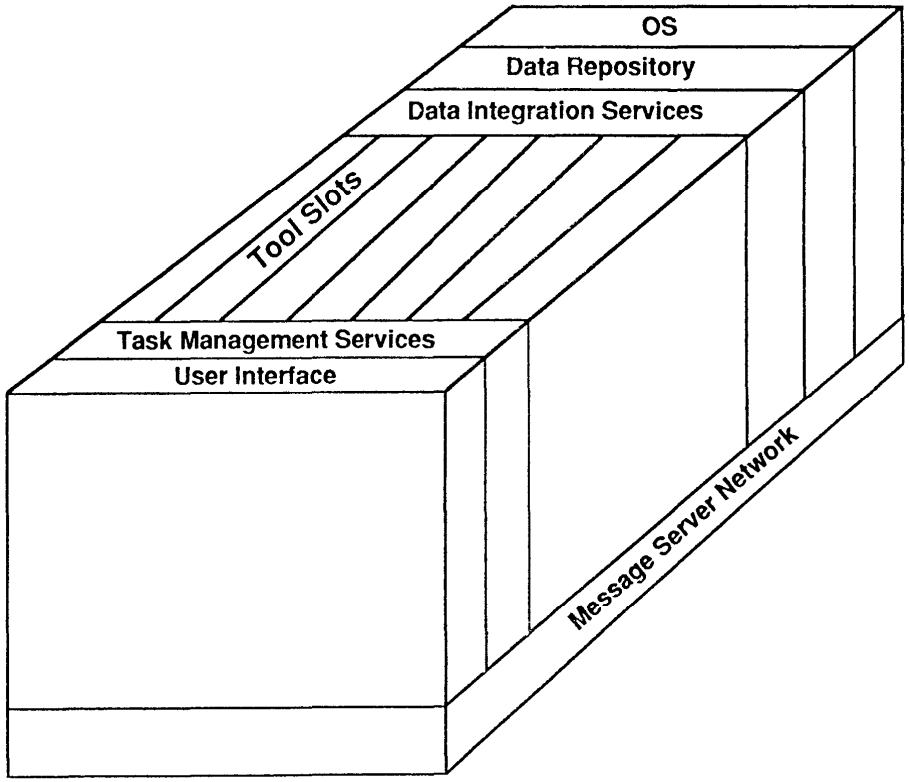


ISM Architecture [Sommerville]

Boxes: any kind of entities
edges: any kind of relations

- "type" of units necessary
- User is not an architecture unit, but the interface to the user is
- Data objects are not units but a component to control these units
- Edges also need a clear static semantics

Another "Architecture": A Standard Built Plan ?



[ECMA]

- No integration: only all tools on the same basic layer working independently

Architectures in Business Administration

- Architecture = Schema e.g. EER
- Translation into relational schema
- Relational schema is a description used at runtime

Database system + description + operations are all means for organization, retrieval, or manipulation of data

Application system using the database systems is not regarded

other modeling approaches: EER, function model, control model used for requirements engineering as well as design

Architectures in UML

- Different diagrams (denoted in different languages) describing different views of the system to be designed
- Relations between diagram languages unclear, and if it were clear, it is still difficult to determine all relations
- "Architecture" is a "refinement" of the requirements specification
- Architecture: one result of a predefined process

⇨ There is no uniform built plan, no clear separation of requirements specification

Our Understanding of Architectures

- Built plan as result of the design subprocess
- Complete plan: all components, relations, and layers
- Consistent to requirements specification, but different from that
- Plan contains all relevant design decisions
- Abstraction compared to realized system (architecture paradigm)
- Independent from programming language, but see the restrictions of that language (e.g. no OO design if system is to be realized in C)
- The plan contains the structure of the system: that part OO, another modular, that layer abstracts from, …
- The plan can be evaluated: portability, adaptability, …
- Architectures are different: application domain, classes of systems, quality of design subprocess, used target system, etc.

Extended Understanding

- Static built plan

- But furthermore:
  - » Dynamic properties: denote typical execution paths
  - » Estimate future system properties: runtime behavior, deadlocks in concurrent systems, satisfiability of time constraints
  - » Describe properties of concurrent systems
  - » Describe distribution aspects
  - » Etc.

⇨ Architecture is not necessarily only a static plan

. . . an abstract plan

"One" Big Plan but Different Abstractions

Graphical overview	design (architecture) diagrams
Textual detailed descriptions	textual MIL language
Different subplans for essential parts	graphical & textual
Different further aspects » Concurrency » Runtime behavior » Semantics of components » Distribution	"annotation" language
Design rationale as part of technical documentation	natural language explanation
"Specification" for use / purpose of modules / subsystems	natural language, decision tables, finite automata

The Languages Have to Express

- Different "types" of components
- Different "types" of composition and usability relations
- Different granularity of parts: subarchitectures and subsystems
- Different further aspects: semantics, concurrency, distribution etc.
- Ideas behind