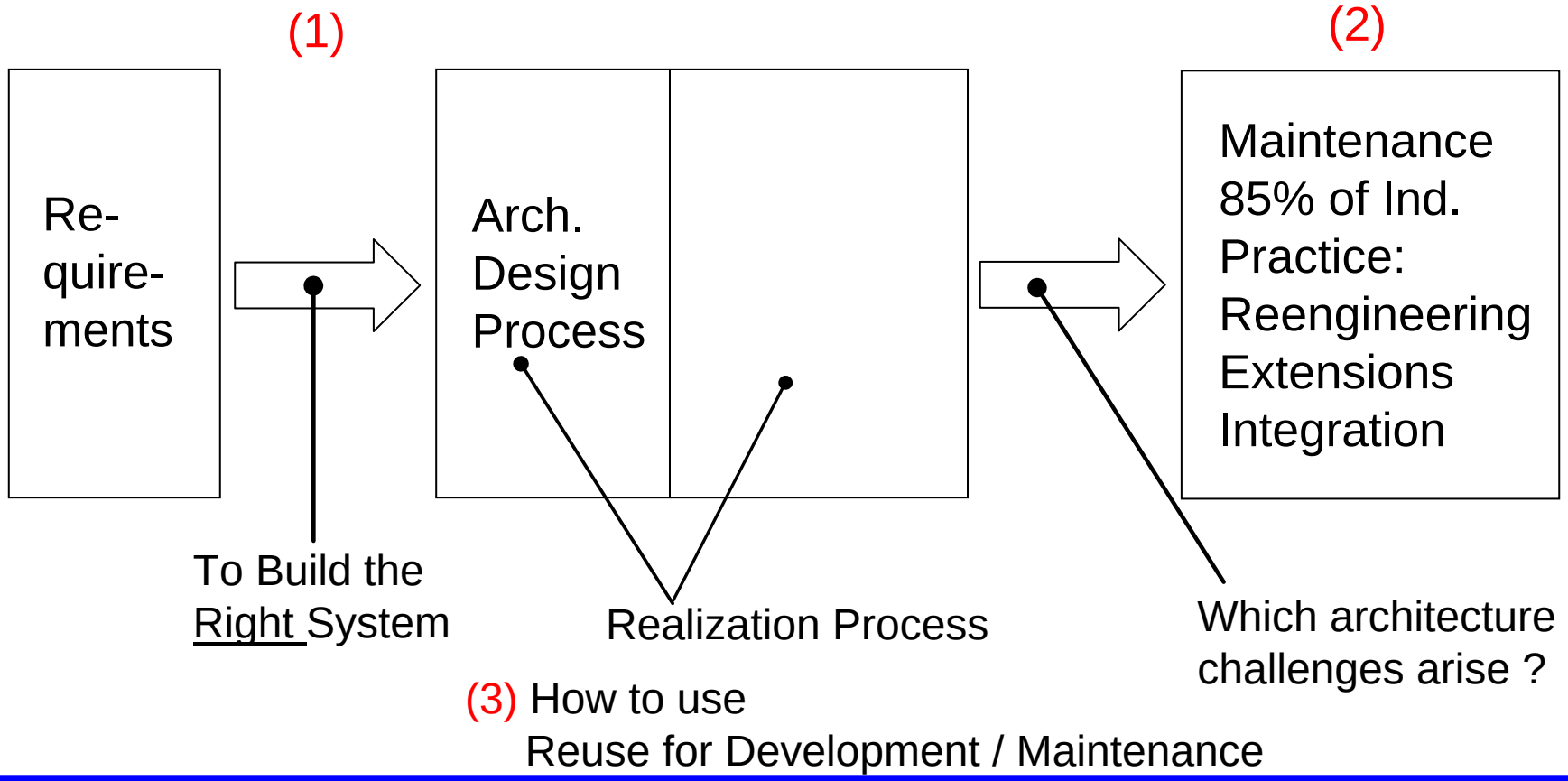
9. Practical Life: Building Right, Maintenance, and Reuse

Aims (1)

- How to do on Architectural Level: shown in chapter 4 – 8
- How to do on Detail Engineering Level (Implementation, Integration, Quality Assurance etc.): assume that we do right
- Three questions remain



Aims (2)

- Give hints, rules, strategies to reduce costs for development / maintenance of software
  by “intelligent“ architectures which
  - » ensure adaptability
  - » make use of reuse
- Summarize what we have learned
- Level of discussion
  - » not modules, subsystems, or complete architectures
  - » but meta-level how to get modules, … , architectures

Contents of Chapter 9

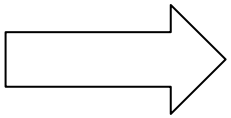
* to be elaborated

Transformation Requirements Specification → Architecture

Requirements Specification (RS)

- Outside Functionality: Functional RS
  - » e.g. system processes application data control
- Nonfunctional RS
  - » efficiency params e.g. user reaction time
  - » restrictions for system to be built
    - storage, runtime,
    - components to be used
    - hardware on which system runs
  - » determinations of later development process

Automatic



Derivation ?

Architecture

- Upper Part in Relation to FRS
  - » main functions
  - » business data entities
  - » controlling UI
- Further Layers and Components
  - » details of data storage
  - » communication structure
  - » basic components

Interactive Architecture Modelling (Design Decisions)

Requirements Specification (RS)

Architecture

Functional RS

via

interactive decisions

Control

"Buisness" Functions and Entities

Further interactive elaborations

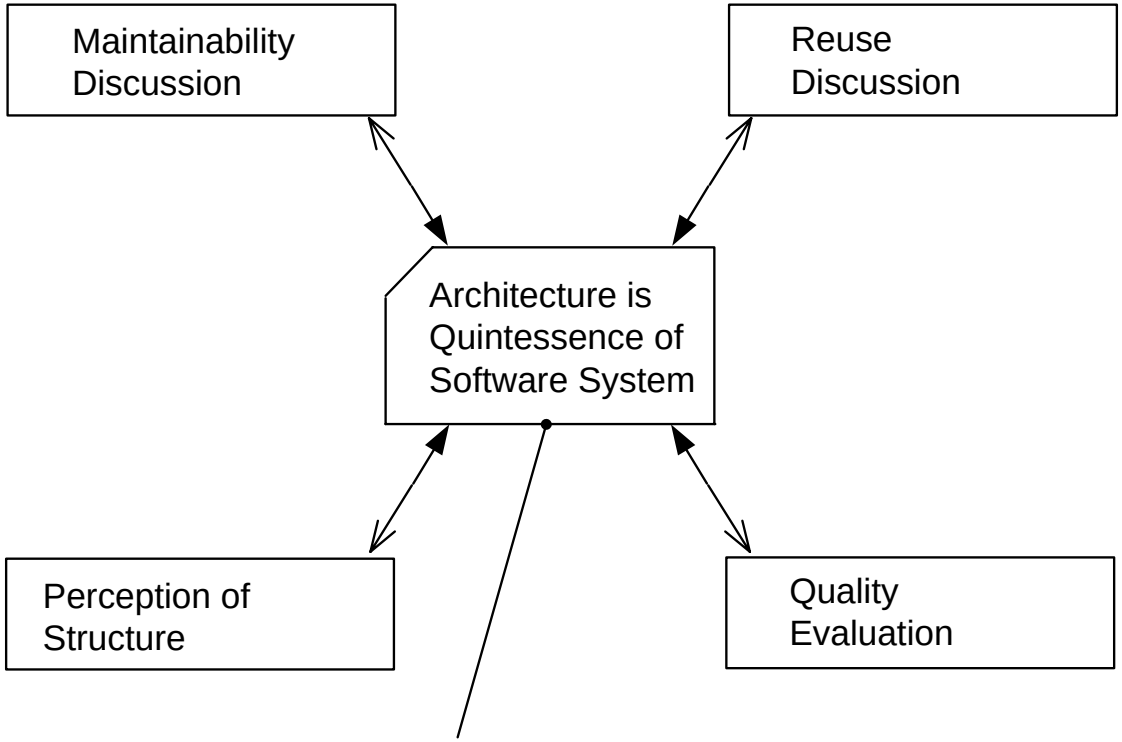
Data Storage

Communication, Distribution

Nonfunctional RS are related to the complete system and, therefore, architecture

Different Purpose and Views !
Architecture: Extensions of RS ?
Automatic Derivation ?

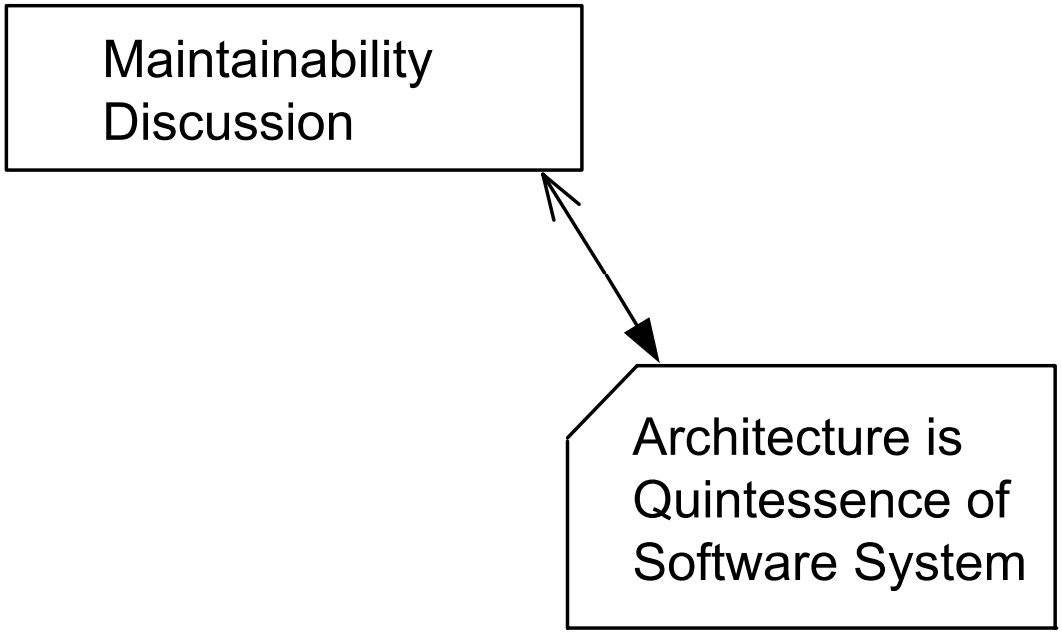
Architecture: Central Structure (1)



not easy to develop
more elaborated if
application
structure class
} knowledge available
needs design rationale, minispecs
readability and clear structure

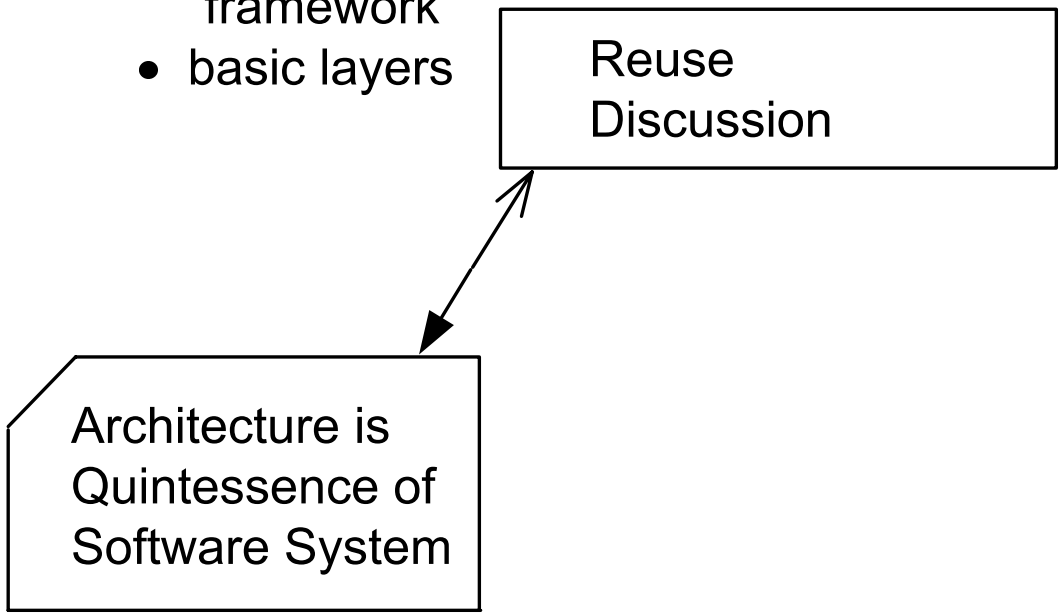
Architecture for Maintainability

- Estimate Effort for Changes
- Determine / Manage Change Chains
- Discuss “What can change ?“
- Detect where changes are forethought



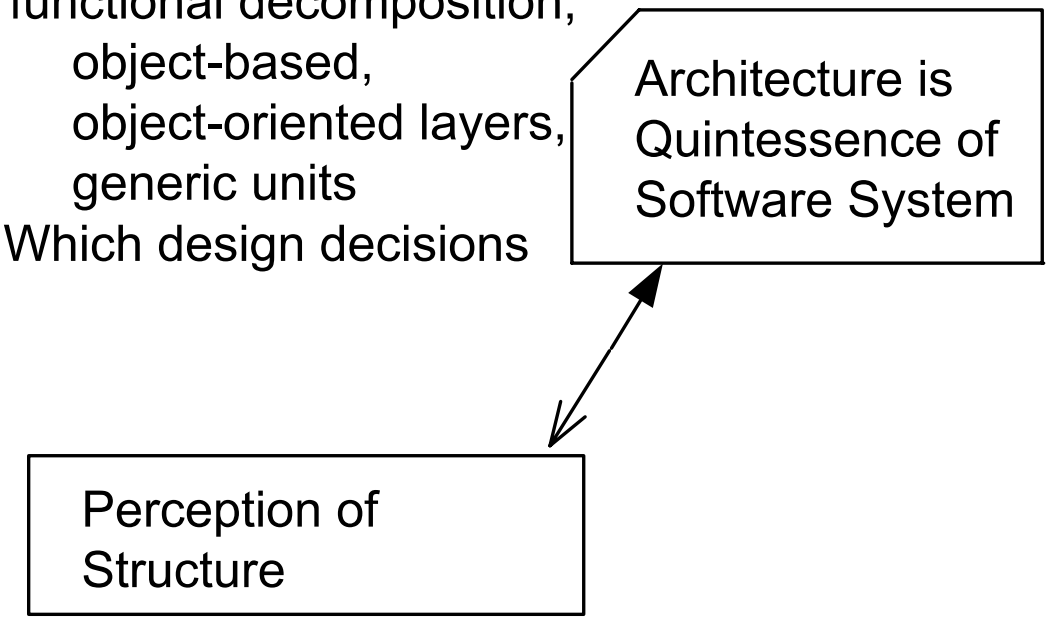
Architecturefor Reuse

- General components: modules, subsystems
- Generic components
- Whole architecture as “framework“
- basic layers

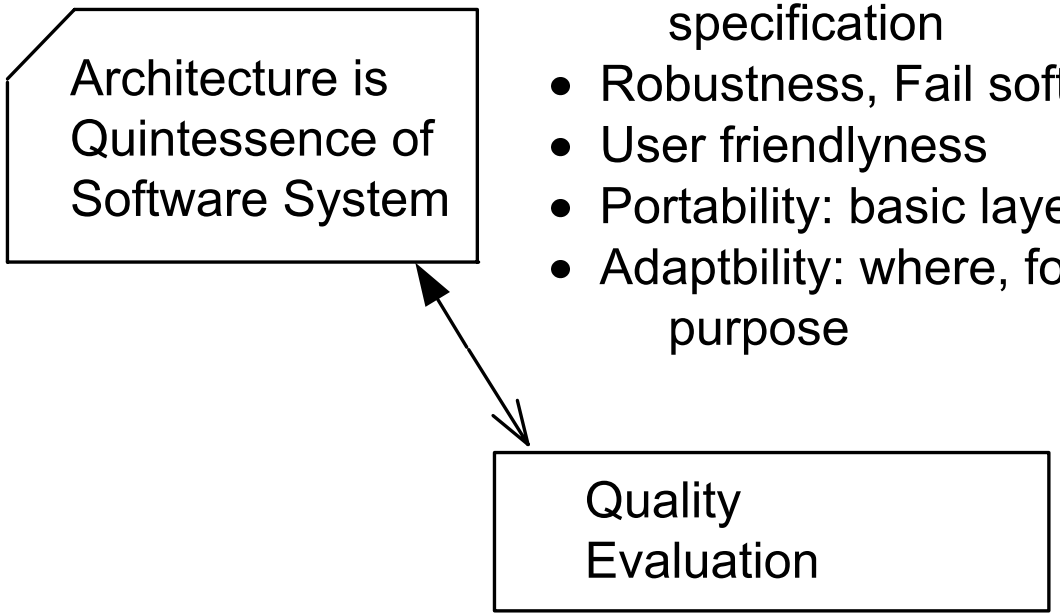


Architecture for Understanding

- Where
  functional decomposition,
  object-based,
  object-oriented layers,
  generic units
- Which design decisions



Architecture and Quality Assurance



- Check against requirement specification
- Robustness, Fail soft
- User friendlyness
- Portability: basic layers
- Adaptbility: where, for what purpose

Strategies for Maintainability & Reuse

Motivation:

- Avoid that we start from scratch every day
- Rules, not exact statements how to act
- These rules can make up a strategy
- Examples:

  general examples

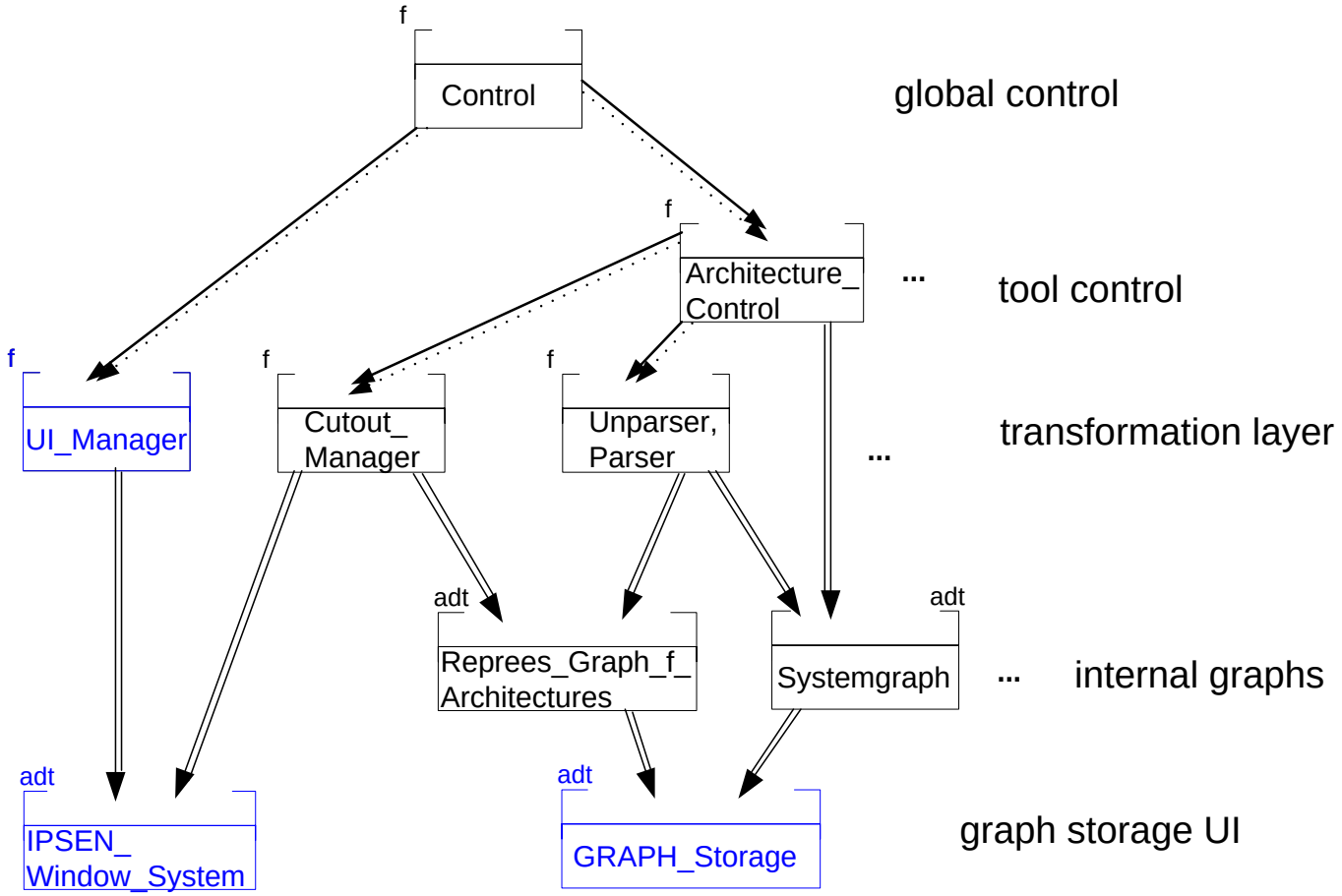
  specific examples from Software Development Environment

Strategy 1: Detect / Offer / Use Common Components

- Are modules, subsystems
- In architecture by general usability
- Different degrees of generality: one system, bound to different systems, one process for different acting systems
- Different users:
  in one subproject, project, department, company, application domain, language implementation, reuse community
- Reuse contribution: trivial
  Maintainability contribution: Part to be developed gets smaller
- No adequate tool:
  - » detection, generalization,
  - » maintenance for component “vendor“
  - » detection, comparison, adaptation, cost estimation for user

Examples Strategy 1

- open source libraries
  language implementation libraries
- SDE

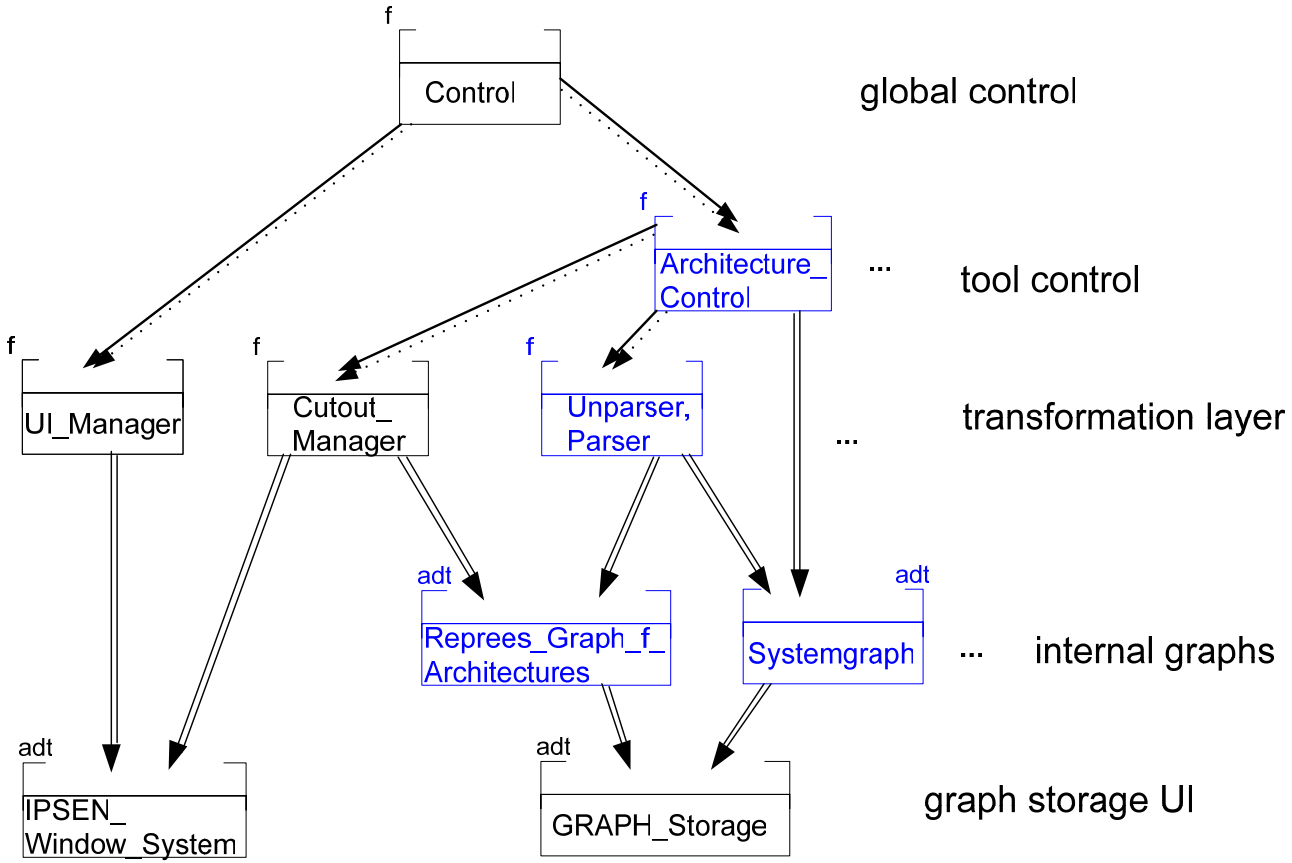


Strategy 2: Software Systems from Building Blocks

- Goal: get system / different variants of a system / essential parts by selecting / “configureing“ building blocks
- Architecture language expressiveness:
  - » subsystems, aggregating further functionality
  - » object orientation
  - » layers of a software system
- Maintenance: add / delete components
  Reuse: components are developed for reuse
- Prerequisite:
  - » deep knowledge about
    - class of systems / application domain
    - availability of corresponding components
    - configuration mechanisms

Specific Examples Strategy 2

- compiler:
  - » with or without optimization / post optimization
  - » different backends for same frontends
- SDE



Strategy 3: Put Changing Parts into Data

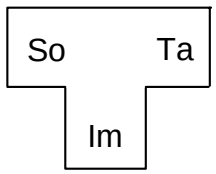
- Data are more easily exchanged than program code
- No contradiction to data abstraction: data are DA applications
- Applicable mostly for strategy 2 components
- New system looks different: Architecture Transformations
  - » error messages not in code but references to entries of a new table
  - » syntax analysis not hardwired but …
- Maintenance Profit:
  - » identity changeable parts and put into ado tables
  - » are easily exchanged
- Reuse Profit:
  - » interpreter on top of table is usable for all tables of a specific form

Strategy 3 Examples

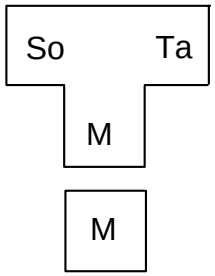
- General
  - » error messages warnings, system messages
  - » layout properties
  - » realization of finite, automata, pushdown automata
- SDE across the whole architecture
  - » messages
  - » possible commands for a logical increment
  - » parser / unparser
  - » layout determination

Strategy 4: Bootstrapping

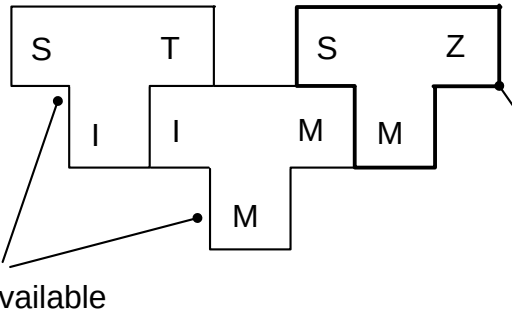
- Used in compiler construction for
  - » language and compiler extensions
  - » porting a compiler to anew machine
  - » compiler improvement
- T notation, execution, composition
  - » so source program, Ta target program, Im implementation language program



T diagrams



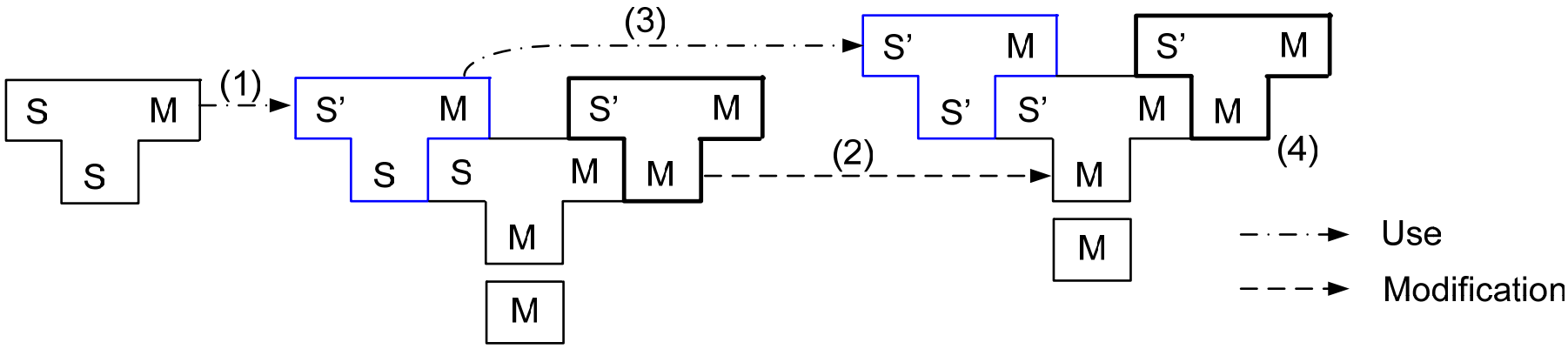
execution



compilation of a compiler

Example Extension of A Programming Language

- Prerequisite
  - » Have $S_M M$ compiler (also bootstrapping)
  - » Have $S_S M$ compiler



Bootstrapping for Language Extension

- Further applications: porting to M´ Ma line,
  - » optimize compiler

Profit & Further Examples

- Reuse: reuse existing compiler structure and code
- Maintenance Profit: above problems are the typical compiler maintenance problems
- More general definition:
- SDE examples:
  - » Architecture of IPSEN and also architecture tool use architecture concepts, languages
  - » This architecture has been used for further tools (mechanical engineering, chemical engineering etc.)
  - » Specification Environment PROGRESS also uses this architecture

Strategy 5: Generate Software Systems

- Motivation
  - » Avoid handcoding
  - » Assumes system → system family
  - » Generation is directed towards specific parts
- Building Block Approach and Bootstrapping are specical cases:
  - » BBA: select & configure
  - » Bootstrapping: “generate“ means modification of compiler basic versions

Different Forms

a) Methodological handcoding
   Recursive Descent Compiler:
   from grammar + method to architecture and code

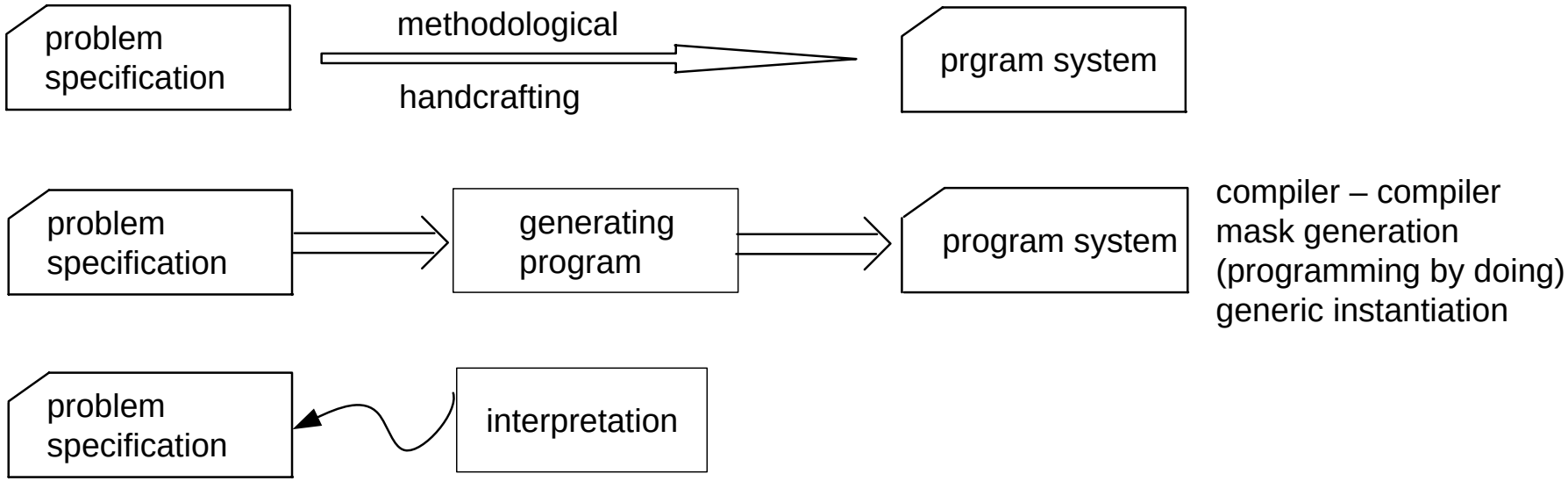
b) Generating a system (codepiler-compiler approach)
   generate program
   or a table

c) Direct interpretation of a specification
   for essential part of a system
   prototyping: can later be improved using a) or b)

d) Genericity for components
   generation is instantiation

e) Programming by doing
   input for automatic translation by PbD

3 Main Generation Categories



Architecture Language Support

- Can express all different forms of generation:
  - » handcrafted gneration: how do architecture transformations look like
  - » how does structure of generating program look like
    - ➢ compiler compiler, generic instantiation program, PbD translation
  - » how does result look like

    table in a system

    generated code in a system
  - » interaction of result and executing device

    interpreter for specification

    driver component for table

Maintenance / Reuse Profit

- Mechanical handcrafting:
  - maintenance by new methodological development process
  - process reuse, eventuelly basic components
- Generating program genericity, PbD translation, compiler compiler
  - maintenance: only new input necessary
  - reuse: of generating program
- Interpretation of a specification
  - maintenance: new specification
  - reuse: interpreter

Strategie 6: Find Similarities

- Motivation:
  reduction of development efforts
  quality improvement
- all above strategies are special cases
  - basic components
  - system by building blocks
  - data into tables
  - bootstrapping
  - generation
- support by architecture language
  object orientation, genericity
  and concepts for strategies 1 - 5

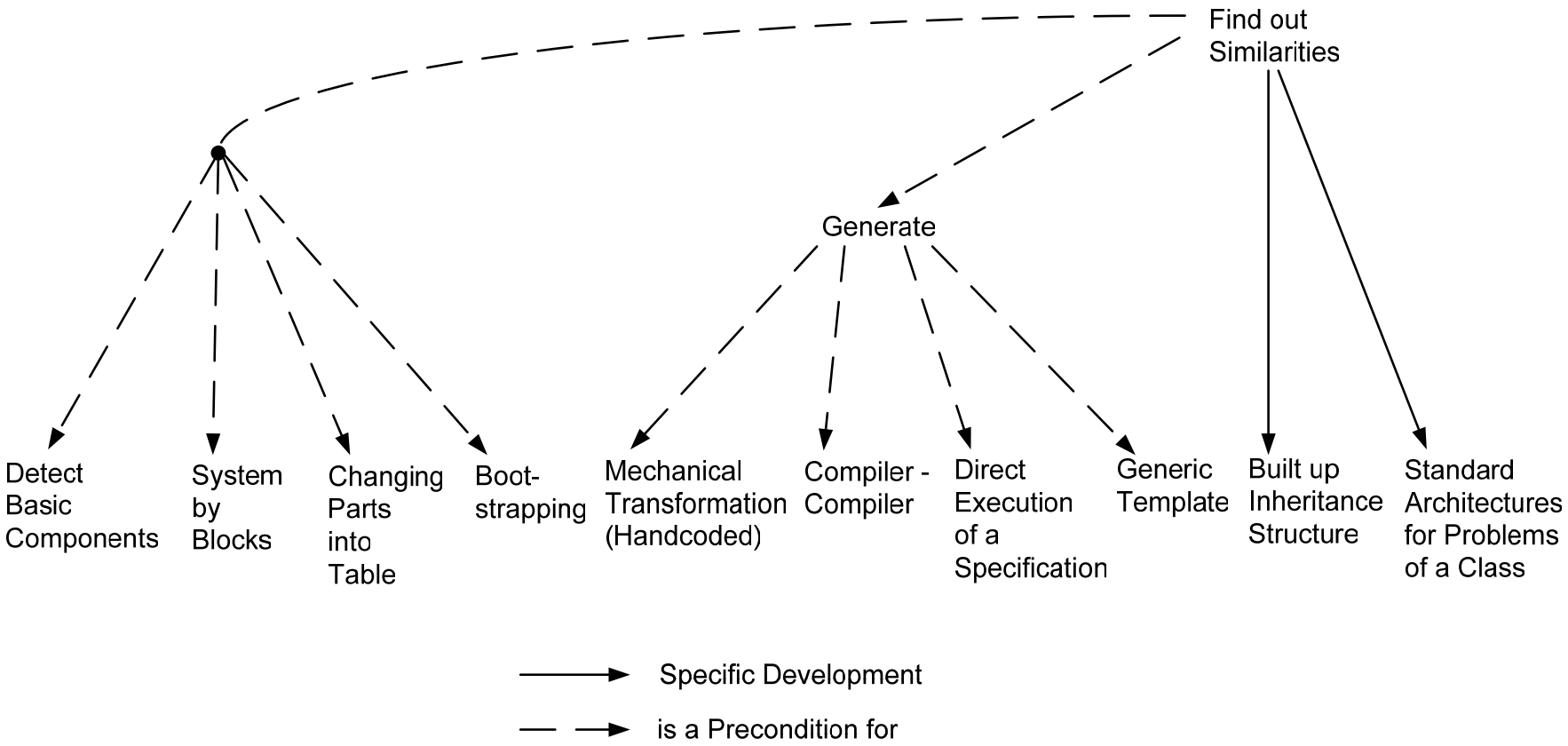
Examples & Profit

- Examples
  - application classes
  - structure classes
  - “standard“ architectures for systems
- Profit
  - for maintenance:
    - similar parts only “built“ once
  - for reuse:
    - similarity avoids new development effort
    - ensures higher quality as similar part is often used

Examples SDE

- Have always same tools:
  editors, analyzers, ...
- Tools have the same external characteristics:
  uniform transformation / command handling
- Graph Specification Engineering
- Similar architecture of tools / environments
- Similar integration schemes
  in one area programming
  between different areas integrated
  overall management
  a posteriori integration, wrapping, distribution
- Bootstrapping
  above
  plus instrumentation → editor operations
  parser → code generation

Summary Strategies



Further Components

- To strategy 1 “Basic Blocks“
  - interpreter / driver on top of tables
  - interpreter system for a specification
- Further “components“ belonging to the development process
  - generic components as template
  - compiler-compiler
  - generic instantiator
  - translation program PbD

  } not belonging to the system under development, belonging to a “tool“ suite