

Component-Based Software Engineering

M.R.V. Chaudron

M.R.V.Chaudron@tue.nl

www.win.tue.nl/~mchaudro/sa2006

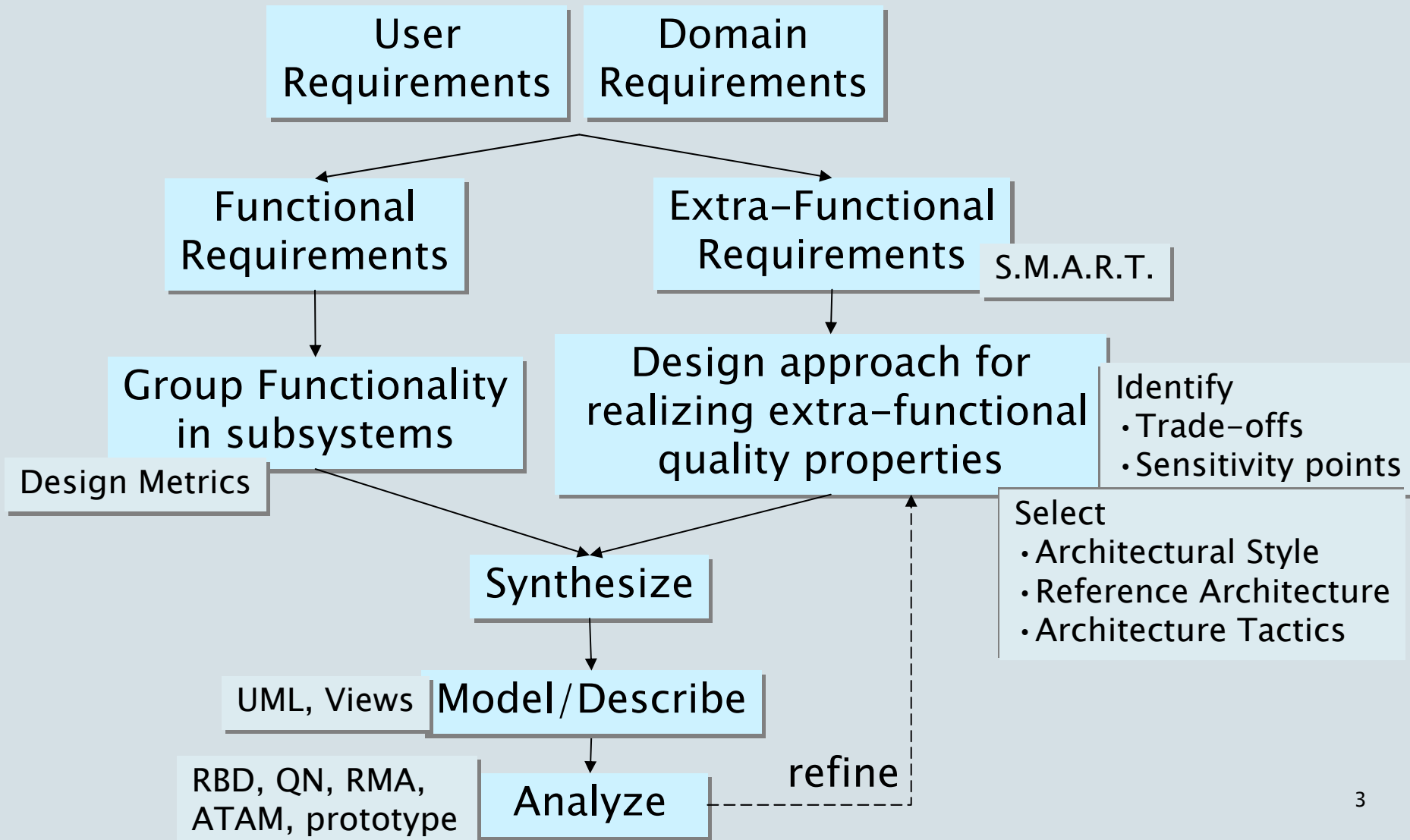
www.win.tue.nl/~mchaudro/cbse2006

Technische Universiteit Eindhoven

Summary of key architecting practices

- Get stakeholder involvement early and frequently
- Understand the drivers for the project (business, politics)
- Understand the requirements incl. quality properties
 - SMART & prioritized
- Develop iteratively and incrementally
- Describe architecture using multiple views
 - abstract, but precise, design decisions & rationale
- Design for change (modularity, low coupling, inform. hiding)
- Analyze in an early stage (use maths! and scenarios)
- **Simplify, simplify, simplify**
- Regularly update planning and risk analysis
- Monitor that architecture is implemented
- Get good people, make them happy, set them loose

Design of Software Architecture



Today's Lecture

- ▶ Logistics about Assignment 4
- ▶ Analytical Approaches for Analysing Architectures
- ▶ Introduction CBSE & Reuse
 - ▶ Motivation, Concepts, Terminology

Assignment 4: Date and Time

- Wednesday, November 8th
- 13:30hrs
- Location: IPO 0.98

Assignment 4: Organization

- Bring your laptop !
- Make sure the tools are working !
- Know how the tools work !
 - Know how the views / functions work
 - Manual
 - Exercises

Assignment 4: Remarks

- Assignment 4 is similar to Midterm Test
- Find out, which group you are in (StudyWeb)
 - Group A: MetricView Evolution
 - Group B: Poseidon / SDMetrics
- Bring your mouse
 - Especially MetricView Evolution users

Component based Software Engineering – Main Sources

Main text:

- Component Software: Beyond Object Oriented Programming
Clemens Szyperski, Addison–Wesley, (*2nd ed*, 2002)
- papers from course web page:
 - www.win.tue.nl/~mchaudro/cbse2006
 - [Volume II: Technical Concepts of CBSE](#), F. Bachman et. al.,
CMU/SEI TR 2000–008, May 2000
 - Components are from Mars, Chaudron & De Jong, WPDRTS
 - Douglas Mc Illroy, 1968/9, NATO Conference on SE

References: Background Literature

- Component-Based Development: Principles and Planning for Business Systems, Katharine Whitehead, Addison-Wesley 2002
Orientation towards business systems.
- Building Reliable Component-based Software Systems, Ivica Crnkovic & Magnus Larson (eds), Artech House Publishers, 2002
Orientation towards technical, embedded, Real-Time systems
- UML Components: A Simple Process for Specifying Component-based Software, Cheesman & Daniels, Addison-Wesley, 2001
Heavily leans on UML, describes some design guidelines for business information systems
- [Building Systems from Commercial Components](#), Kurt C. Wallnau, Scott A. Hissam, and Robert C. Seacord, Addison-Wesley (SEI series), 2001

Observations on the practice of SE

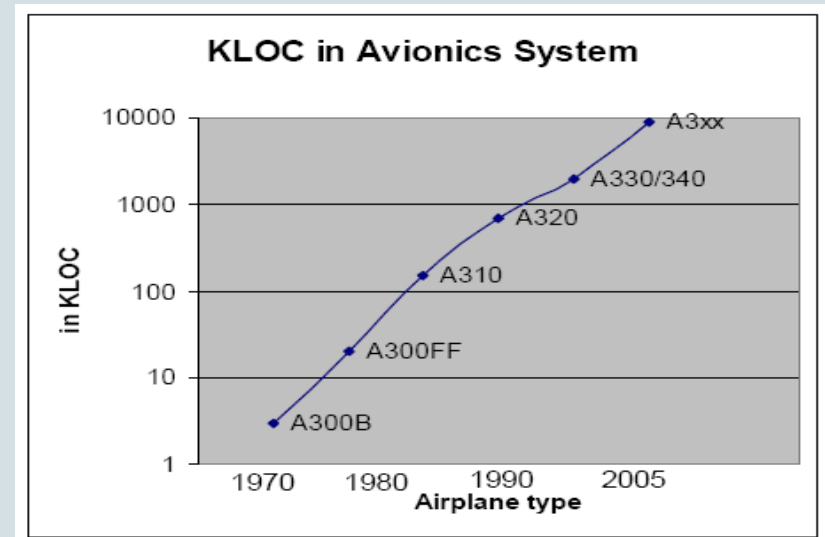
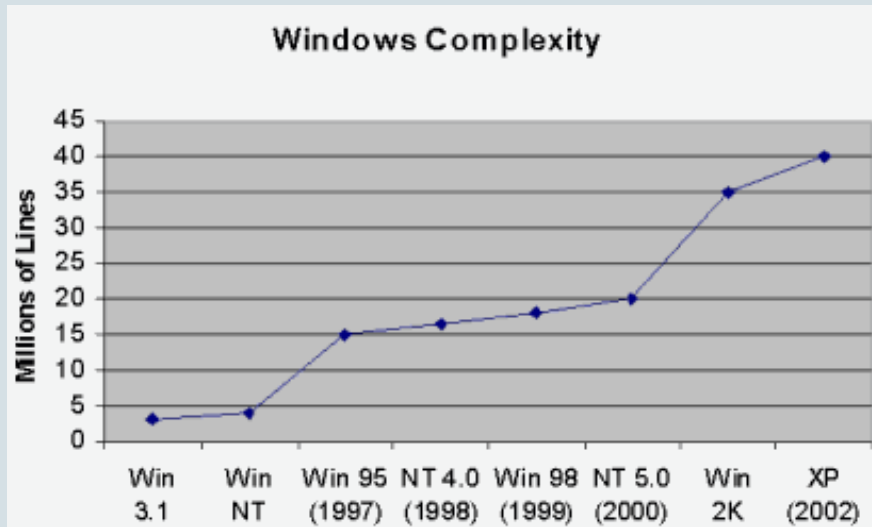
About 80% of software engineering deals with changing existing software

It is not the strongest of the species that survive, nor the most intelligent, but the ones most responsive to change.
-- Charles Darwin

Time to market is an important competitive advantage: incorporate successful innovations quickly

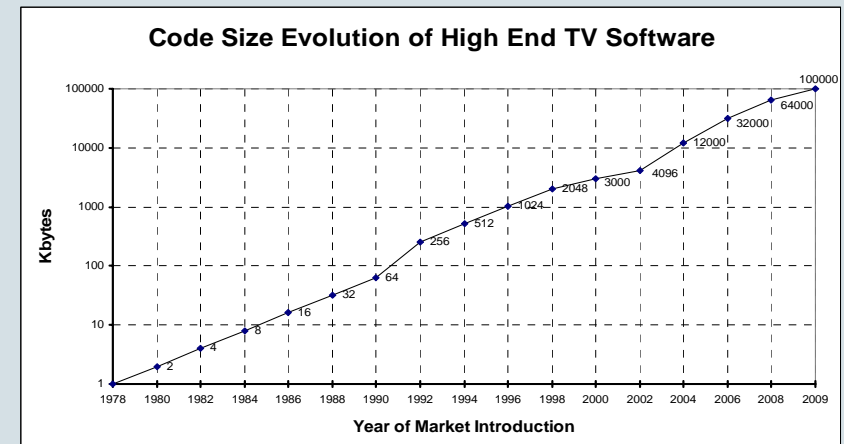
- Systems should be built to facilitate change
 - easy removal and addition of functionality

Increasing amount of software in systems



Nb: logarithmic scale

The Amount of software increases



Problems of Software Engineering

- The size & complexity of software increases rapidly
- The time-to-market must decrease significantly
- The cost of products must be reduced
- Single products become part of product families
- Software is upgraded after deployment

Productivity & Flexibility

CBSE is part of the solution, but not in isolation

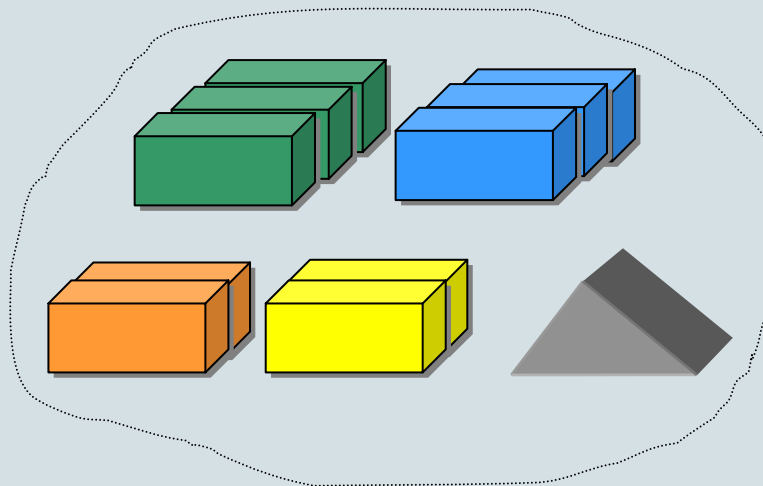
The CBD-‘Solution’

Systems should be assembled from existing components

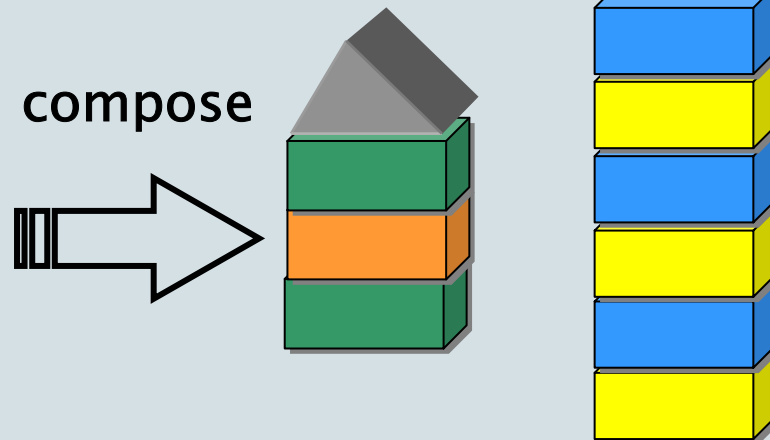
Idea dates (at least) to the 1968 NATO Conference

Douglas McIlroy: Mass Produced Software Components

component repository & market



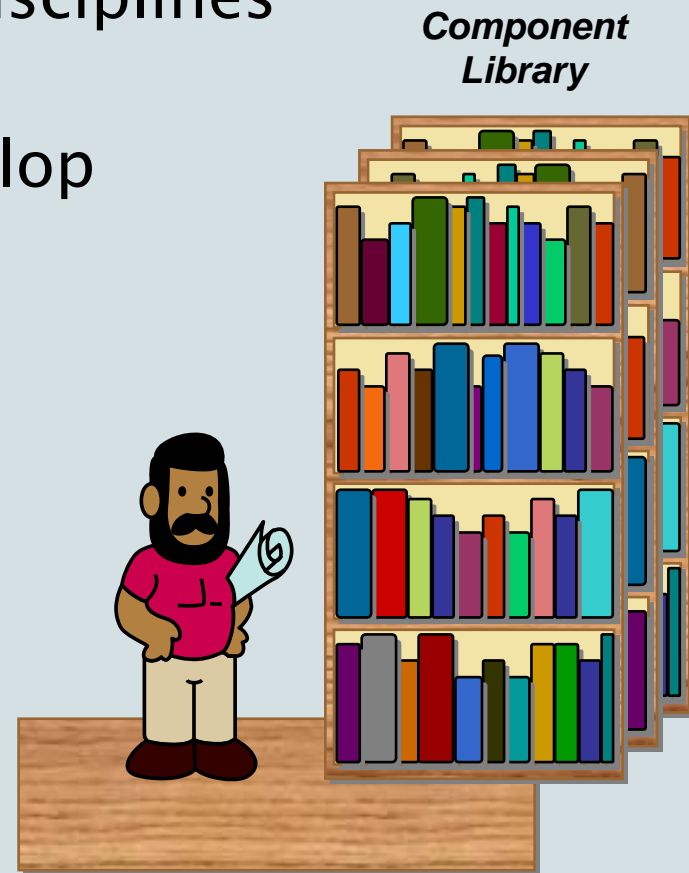
component-based systems



Why Components?

Following other engineering disciplines (civil and electrical), software engineering is looking to develop

a catalogue of
software building blocks
& connection standards



Confusing or helpful?

What is CBSE?

based on definition of SEI in CMU/SEI-2000-TR-008

Component-based Software Engineering is concerned with the *rapid assembly* and *maintenance* of component-based systems, where

- components and platforms have *certified properties*
- these certified properties provide the basis for *predicting properties of systems* built from components.

Predictability is a *key* property of mature engineering disciplines. It enables feedback on design and adaptation; i.e. development time is reduced because we can analyze prior to building

Questions of the CBSD approach

- How to separate the development of components (from each other and from the overall system)?
- How does development and evolution of one component of a family of systems impact other components?
- A component is *almost* a perfect fit for a new system. Can the existing component be extended in unanticipated ways without touching the source code?

Questions of the CBSD approach

- How can one tell if two components can work together? Will a combination of components satisfy performance requirements?
- How will one component find another component at run-time?
- Which instance should be used if multiple versions of a component are present at run-time?
- How can component that run on different platforms interoperate?

Motivations for CBSE

- Productivity
- Quality
- Time-to-market
- Maintenance

Strategic business goals that increase

- Turnover
- Market share

reduce

- Cost of development
- Cost of ownership

€ € €
profit
€ € €



Business Drivers for CBD



Improve Productivity:

Build more software using fewer resources through enabling the assembly of systems from components that may be independently developed by different parties.

CBSE & Software Productivity

Increase competitiveness (sw/€):

- Reduce cost of development
- Increase software/€

Limited human talent (sw/people):

- Increase software/person
 - ⇒ reuse existing solutions, rather than invent them



CBSE & System Quality

Improve Quality:

Idea: Assuming that a collection of high-quality components is available, assembling these should yield systems of high-quality.

1. The cost of establishing the high quality of components is amortized over multiple use.
2. Multiple use of a component increases the likelihood of finding and removing errors.

Quality: *ilities – Technical Drivers



CBSE may help improve system qualities

CBSE & Maintenance

The use of CBD requires good modular design.

This modularity provide quality properties like

- comprehensibility/understandability
- maintainability
- flexibility
- ...



CBSE & Time-to-market



If the reuse of a component requires less time than the development of a component, systems can be built faster.

Reuse-based Software Engineering

Reuse-based SE has many business drivers in common with CBSE:

- increase productivity & quality
- reduce time-to-market,
- reduce development cost

However, reuse imposes less technical- and design-constraints on the unit of reuse (*asset*).

CBSE enables Reuse, Reuse is not sufficient for CBSE.

Reusable Assets

Virtually any product of the SE process can be reused:

- Requirements
- Architectures
- Designs
 - design patterns, interfaces
- Source Code
 - ranging from to libraries, patterns, to modules, to macros, coding conventions, ...
- Test Scripts

What is a software component?

How can you recognize a software component?

Suggestions from the audience?

Reflect on differences between civil and electrical engineering on the one hand and software engineering on the other hand

What is a Software Component?

Probably more definitions than for ‘software architecture’

A software component is a unit of composition with contractually specified *interfaces* and *explicit context dependencies* only

A software component is *independently deployable* and subject to *composition* by third parties.

Clemens Szyperski, 1997

Separation of interface and implementation

What is a Component?

A reusable software component is a logically cohesive, loosely coupled module that denotes a single abstraction.

Grady Booch,
Software Components with Ada, 1987

Tries to provides some design guidance.

What is cohesive? loosely coupled? single abstraction?

What is a Software Component?

- a unit of
 - independent production, acquisition, deployment, and maintenance
 - replacement
 - reuse
 - composition

} properties depend on how they are used
- a package of cohesive services
 - encapsulates design decisions
 - explicit dependencies
 - cohesive / denotes a single abstraction
 - generic (application independent)
 - configurable

} *intrinsic* properties
- loosely coupled
- standardized interfaces
- self-contained

} *context / system* properties

What is a Component?

“A binary unit of independent production, acquisition, and deployment that interacts to form a functioning system.”

– C. Szyperski, *Component Software*

“A component is an independently deliverable package of operations.”

– *Texas Instruments Literature*

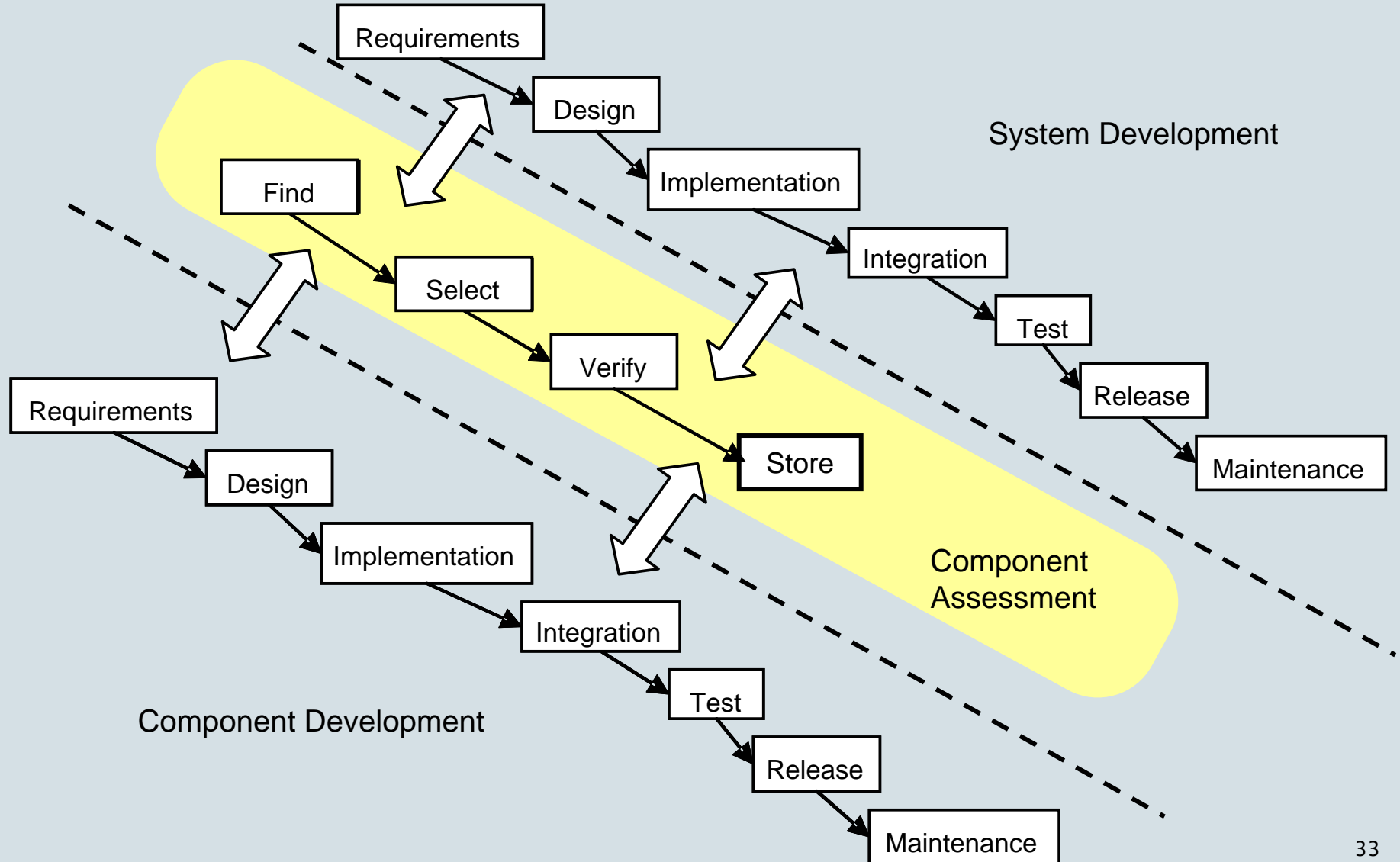
“A replaceable unit of development work which encapsulates design decisions and which will be composed with other components as part of a larger unit.”

– Desmond D’ Souza, in *Catalysis*

What is composition?

- What is composition?
- What things can hinder successful composition?
- How does composition depend on the ‘matter’ of a component?

Component assessment process

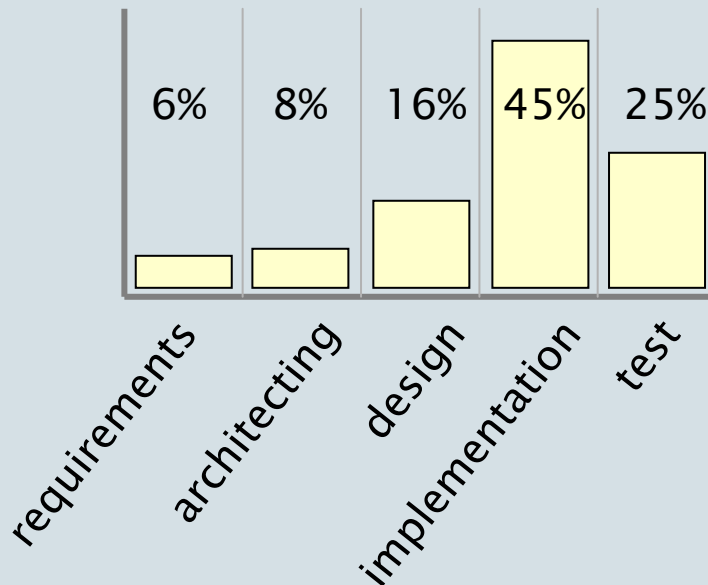


Expected Effects of CBD on the SE Process

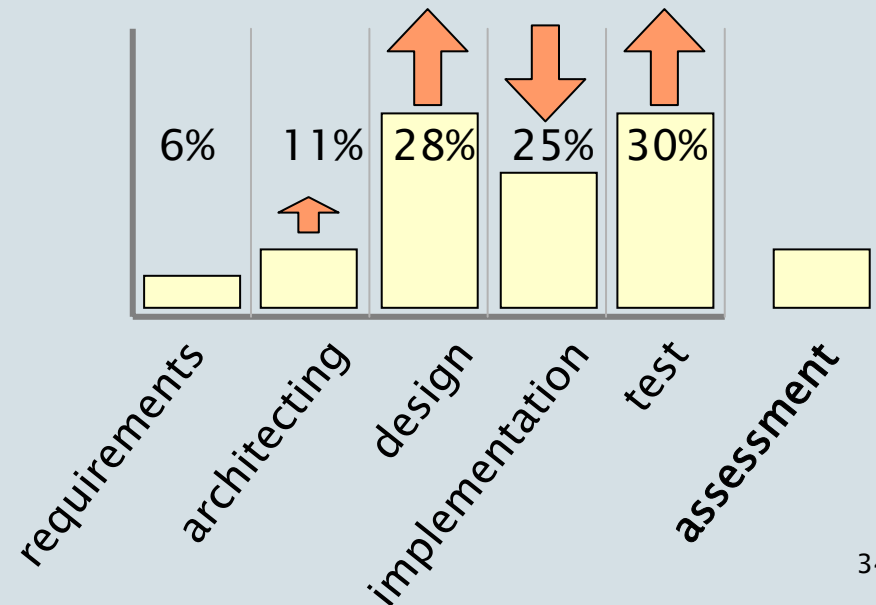
Shift in effort:

- reduction in implementing 40–50% (assuming black box reuse)
- increase in architecting (10–15%)
- increase in design (15–60%)
- increase in testing (15–20%)

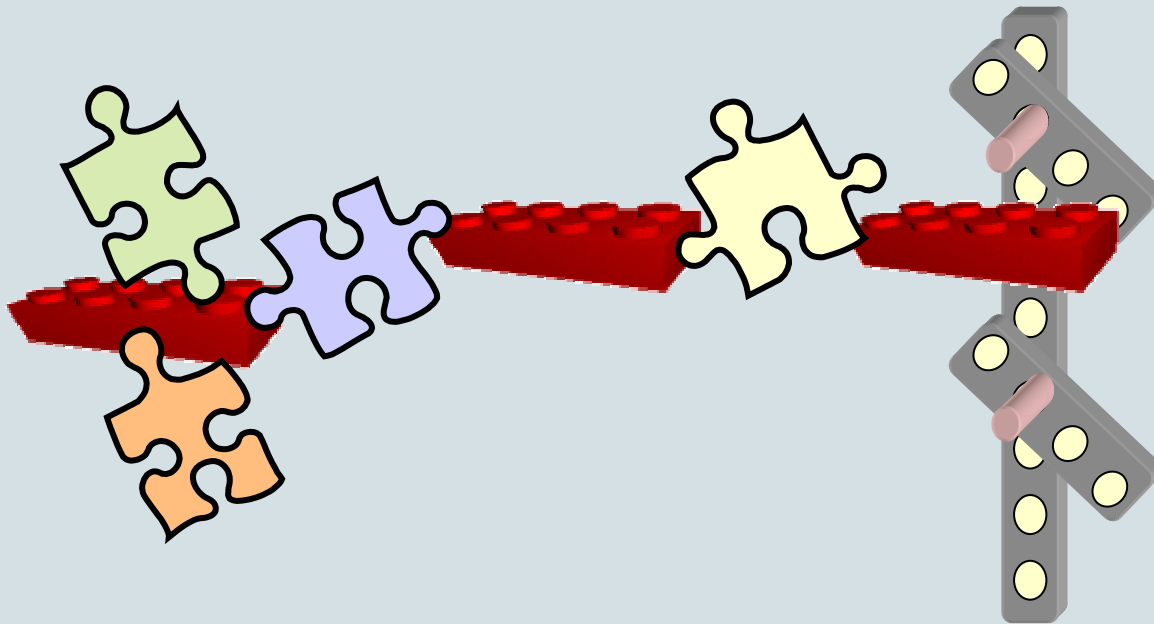
Typical effort distribution



Expected distribution for CBD

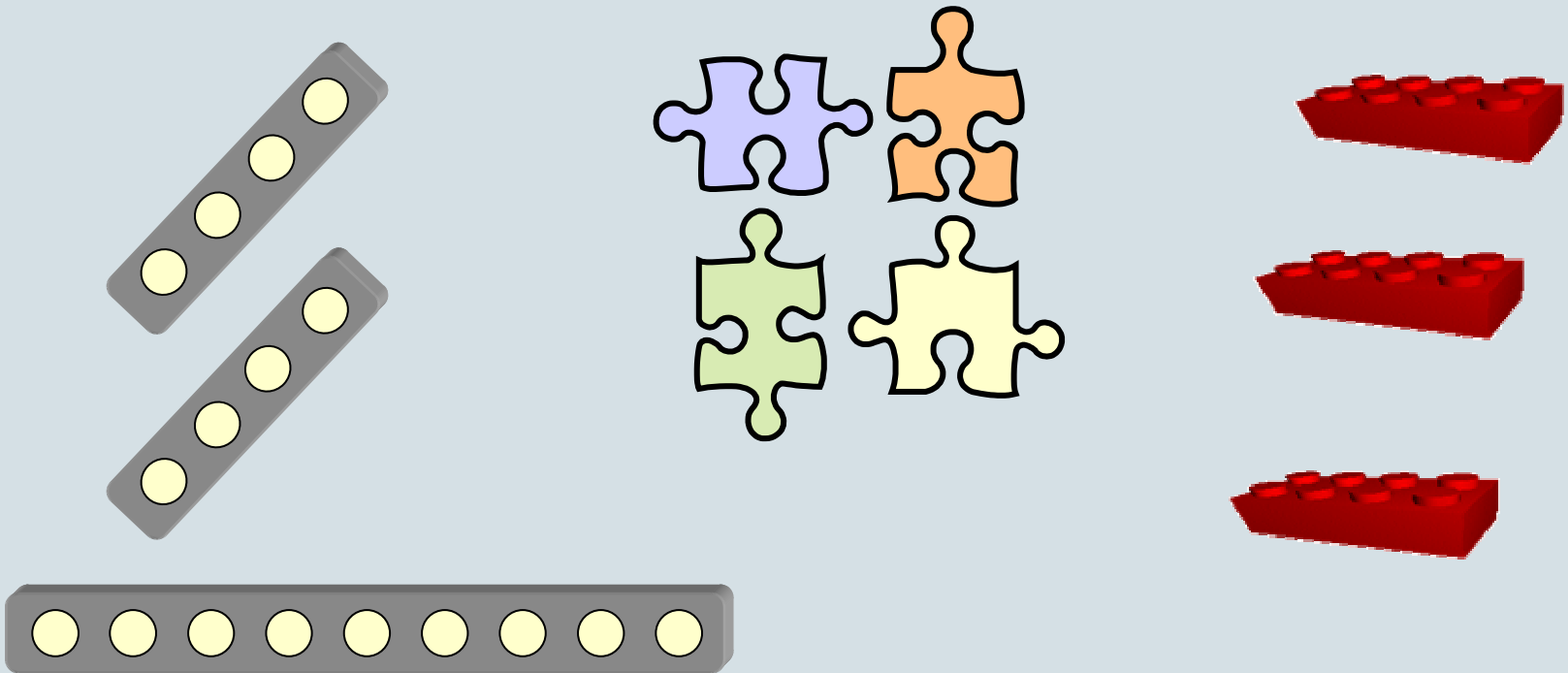


CBD in Practice



Lego + Fisher Technik + Meccanno + Ministek + ...

A component can be used within the scope of a component-model



.Net, Enterprise Java Beans, Corba Components + ...

Component Model

Definition A *component model* specifies the standards and conventions that are needed to enable the composition of independently developed components.

Different types of requirements

- When to compose?
 - At what stage is flexibility needed?
 - Design–, compilation–, run–time
- Different extra–functional requirements
 - Performance, security, reliability, ...
- Different facilities
 - Transactions, real–time scheduling, ...

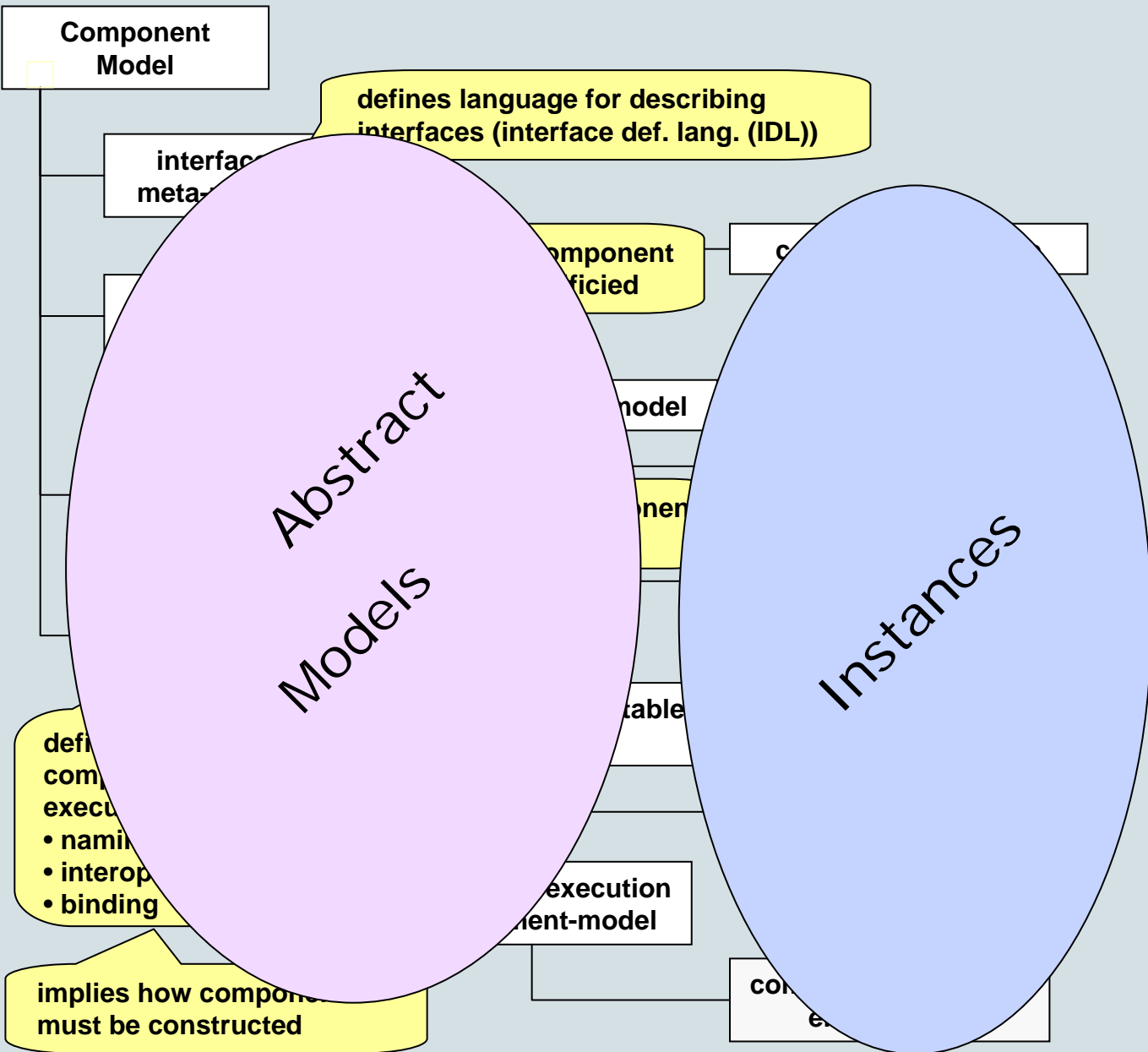
Aspects of Component Models

A component model is a set of agreements that is needed to enable the *combination* of components.

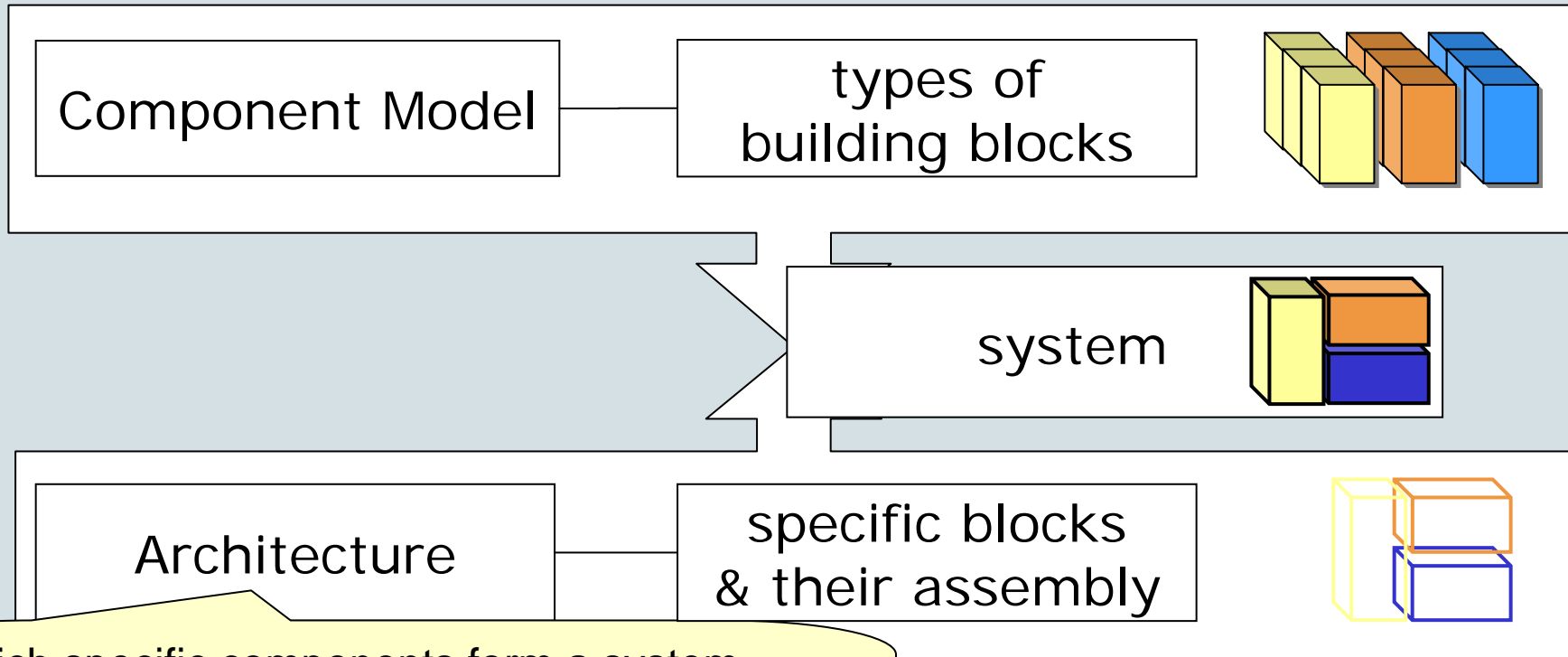
A component model typically addresses

- Life-cycle management: instantiation, (de)activation, removal
- Binding mechanisms
- Interaction style
- Data exchange format
- Process model

Related: Packaging Model



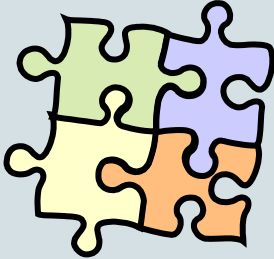
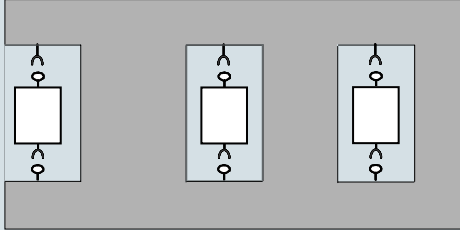
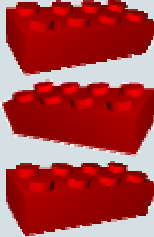
Component Model & Architecture



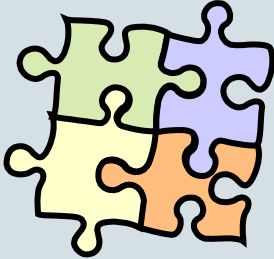
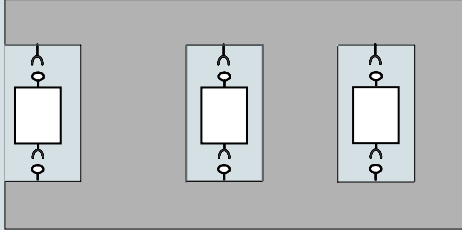
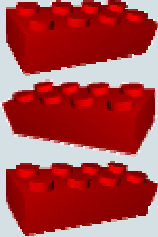
- which specific components form a system
- which specific interfaces these components have
- specific patterns of execution

System Quality properties are determined by both components, component model & architecture

Architecture vs. Generic Components

Architectural component	Framework component 'plug-in'	Generic component
 <p>there is exactly one type of functionality that fits each place in the system</p>	 <p>specific types of function fit at a particular place in the system</p>	 <p>functionality can be combined arbitrarily</p>
independence of component design		
Design of components is coordinated	Design of components is scoped	Design of components is independent
Extensibility of component		
Extension by architecture only	Internet-browsers	
example		
Architecture XYZ	Internet-browsers	Unix' Pipe & Filters

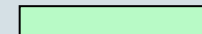
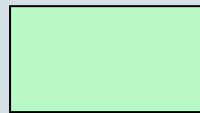
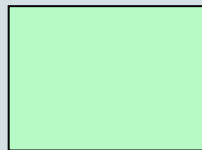
Architecture vs. Generic Components

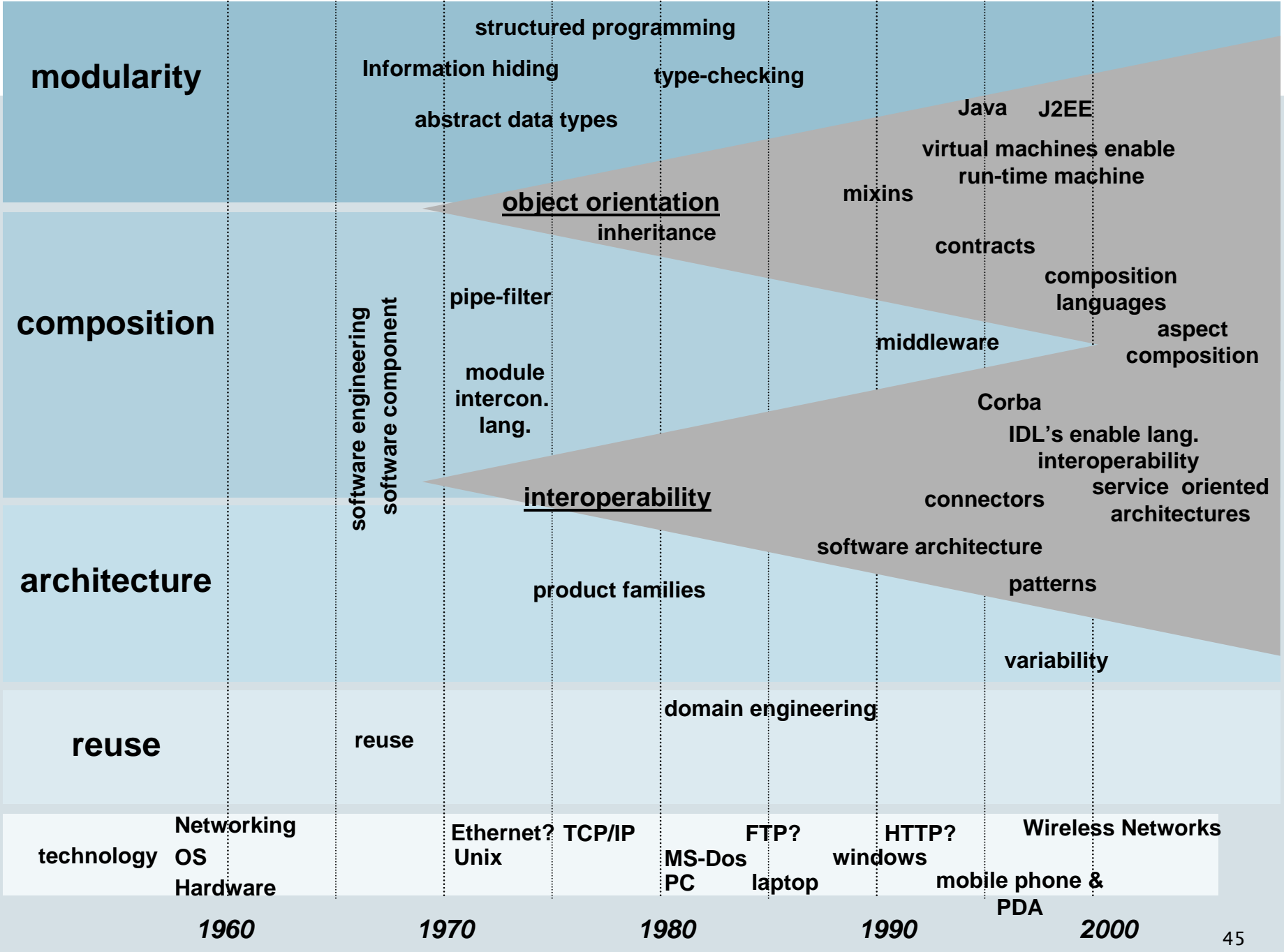
Architectural component	Framework component 'plug-in'	Generic component
		

relative contribution to component model

architecture
driven
features

basic inter-
operability





Milestones in CBSE History

CBSE builds on a number of trends; each of these trends contributes to increasing the *independent treatment* of components in different stages of software development.

Independent compilation of modules

Independent design of modules

Independent loading of modules

Independence from programming language – IDL

Independence from distribution – middleware

Decoupling function from implementation – separation of concerns
ADT & Object oriented programming

Decoupling function from application – service oriented architecture

Decoupling data from data structure – XML



Decoupling

Summary

- CBD aims to improve
 - Productivity, Quality, Time-to-market
- CBSE is about the composition of independently developed components
- Composition is facilitated by a component model
- CBSE enables Reuse; Reuse is about assets

Questions?

You should know

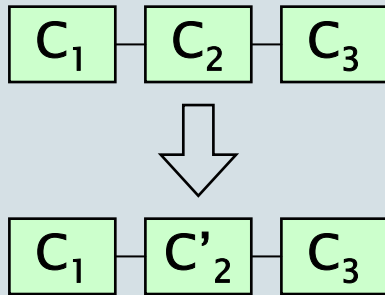
- an answer to ‘What is a component?’
- what a component model is
- the relation between reuse, CBD

Self-Study material:

- Tech. Report SEI: Technical Concepts of CBD

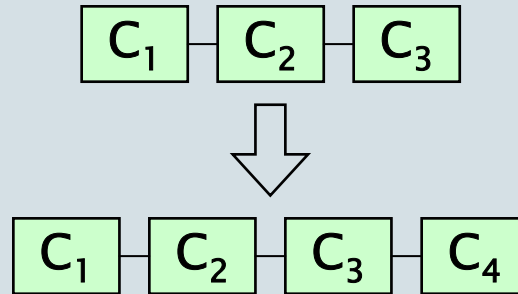
Different usage requirements

Substitutability



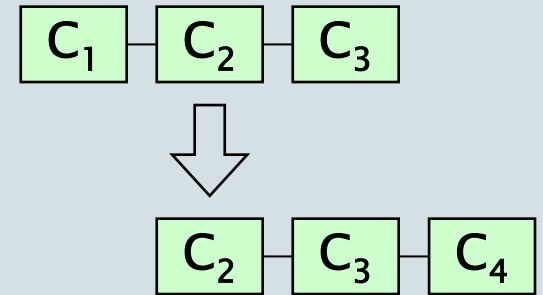
complete specification

Extensibility



extensible architecture

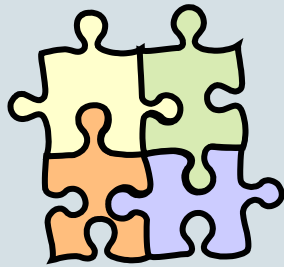
Decomposability



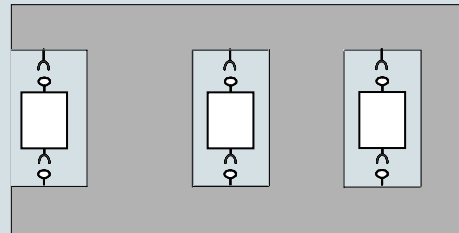
generic components, flexible architecture

Architecture vs. Generic Components

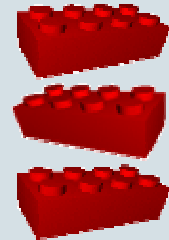
Architectural component



Framework component
'plug-in'



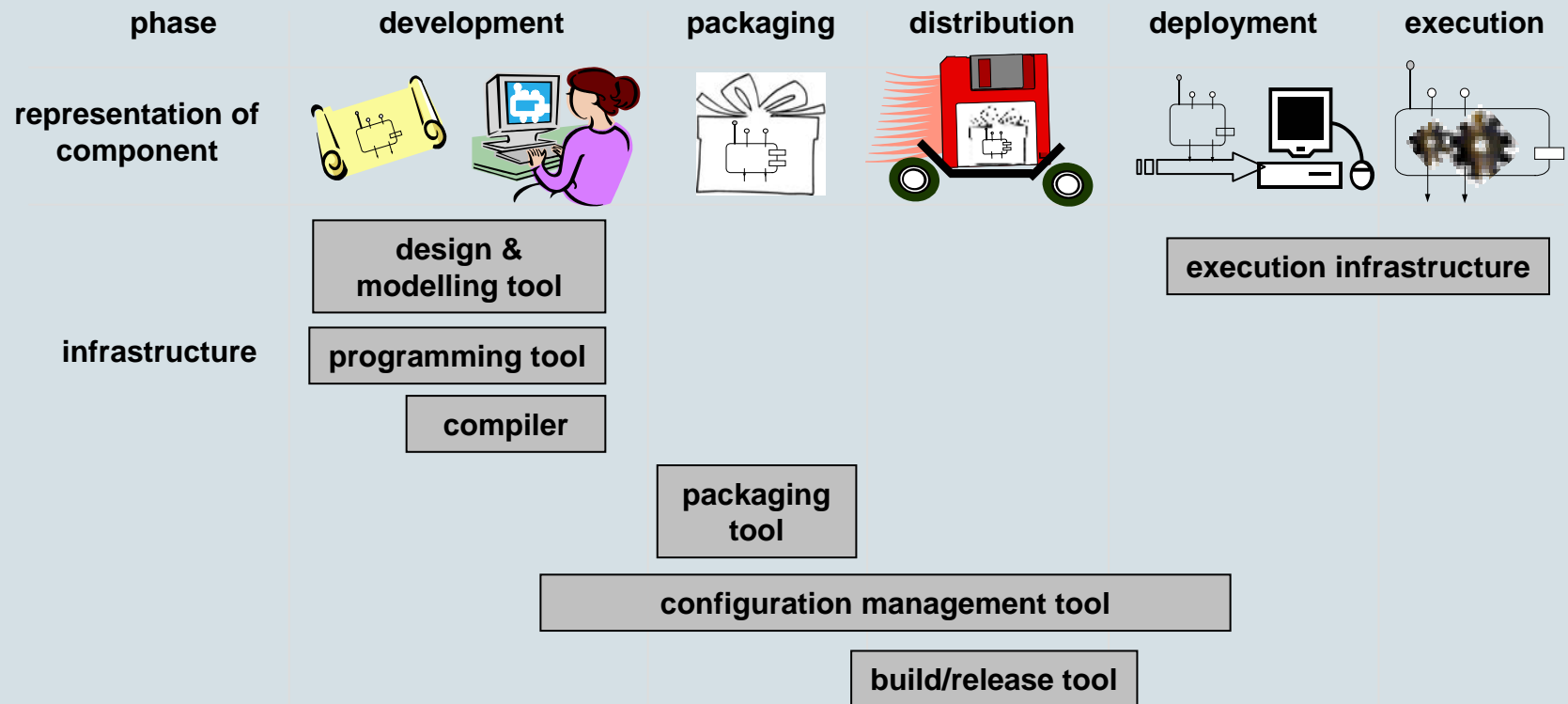
Generic component

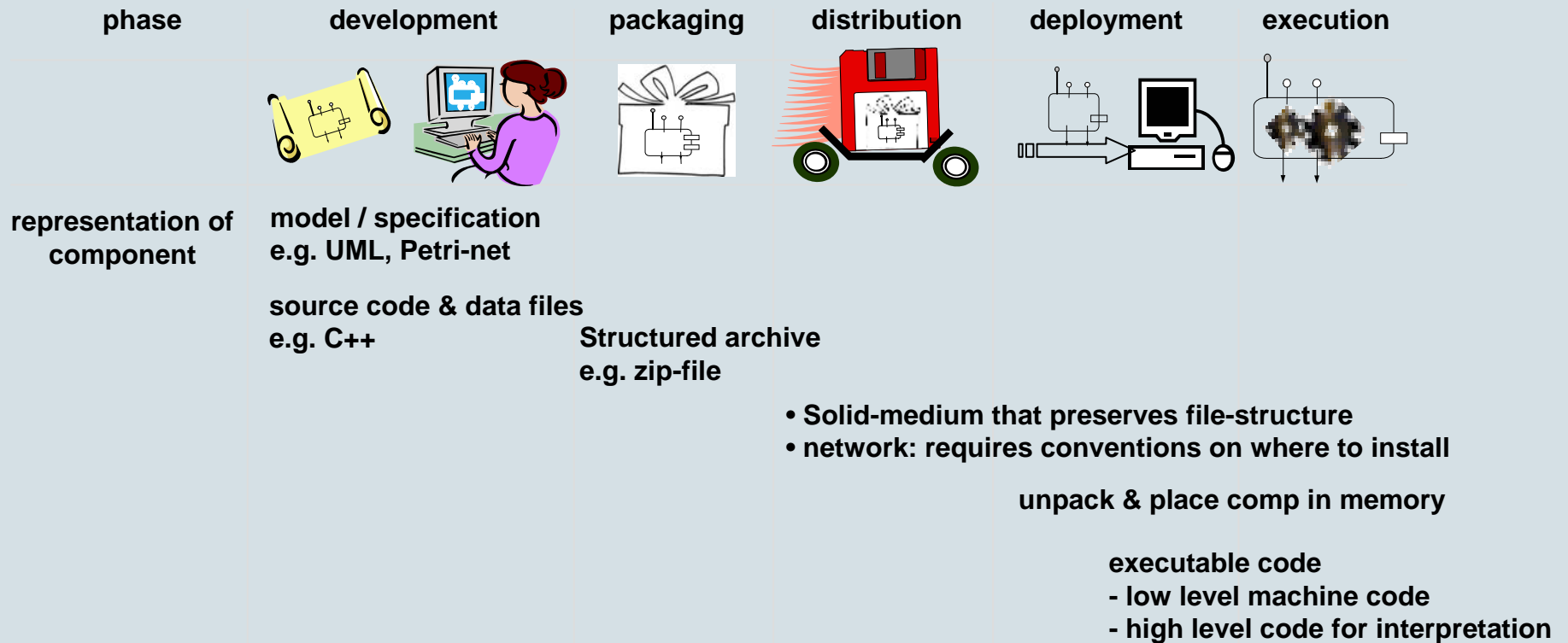


there is exactly one component that fits each place in the system

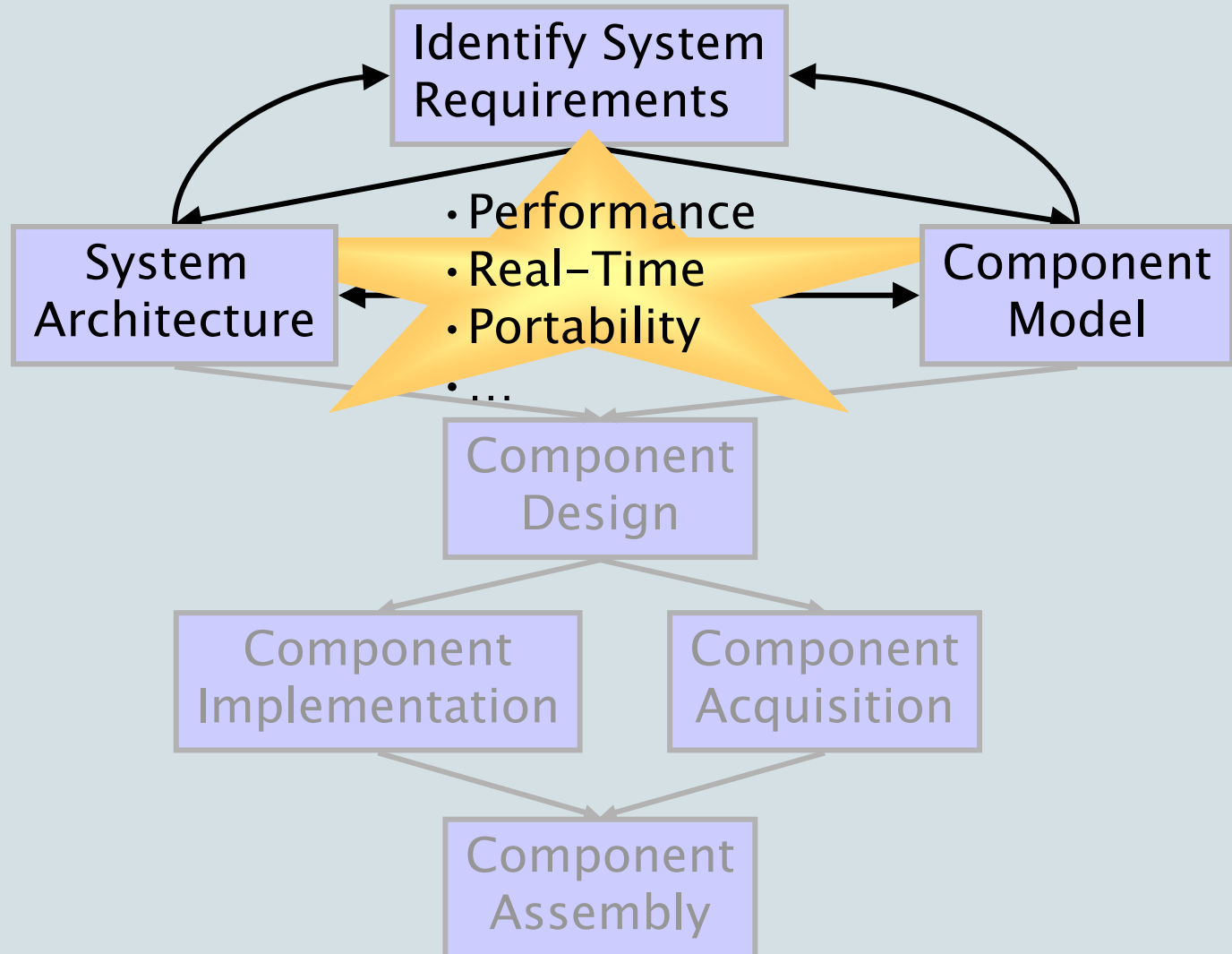
several components fit at a particular place in the system

every component fits at every place

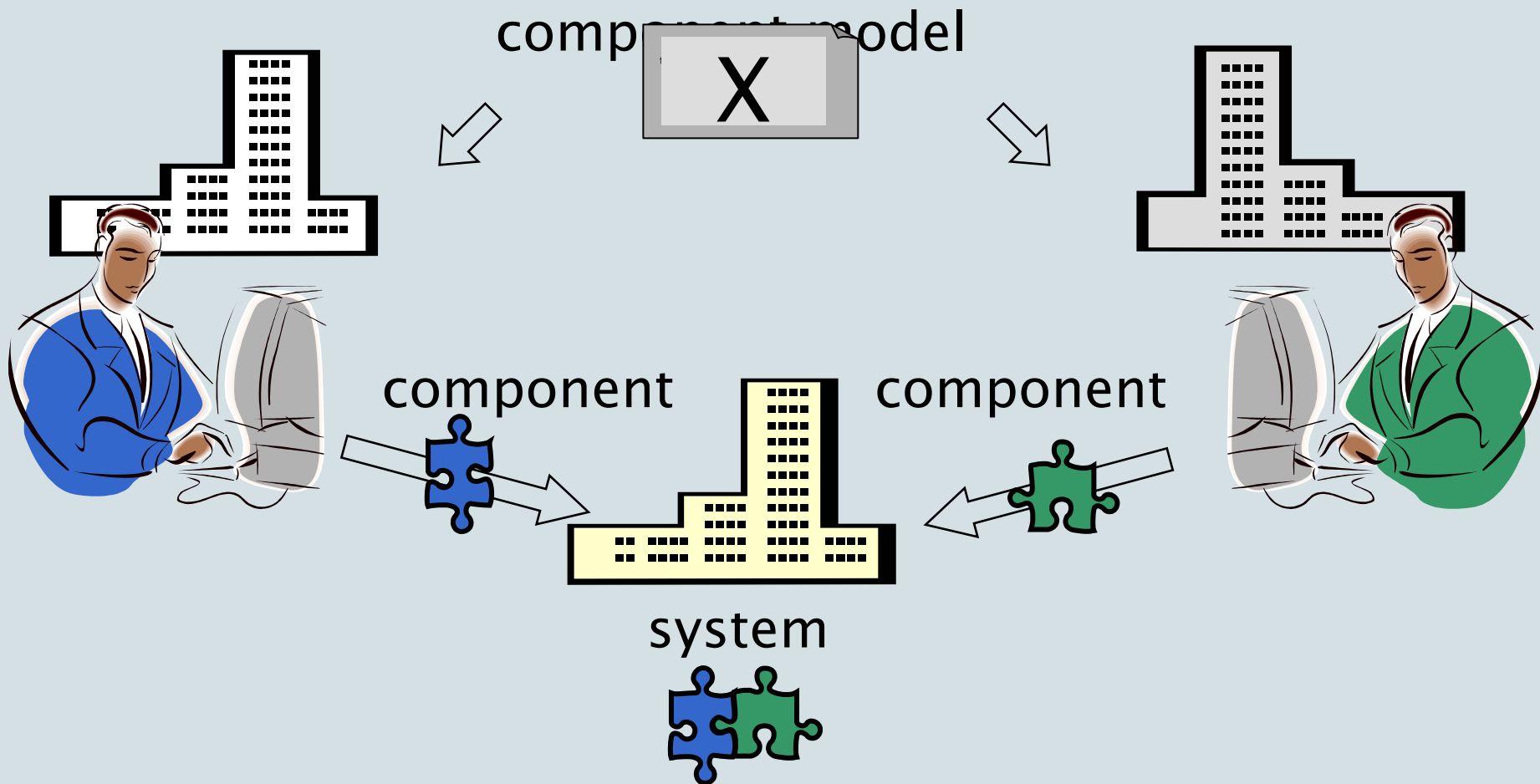




CBSE Development Activities



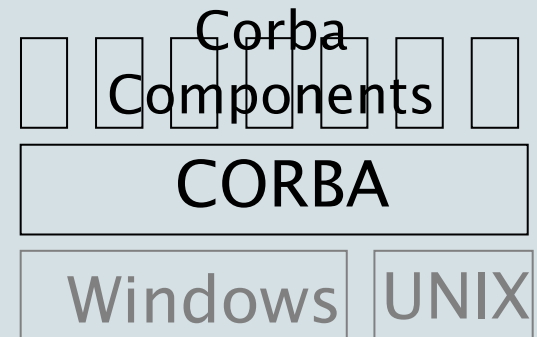
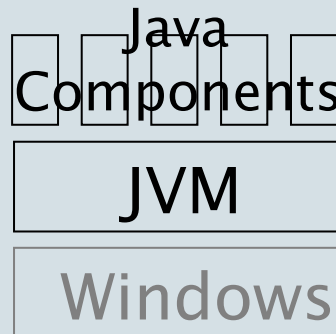
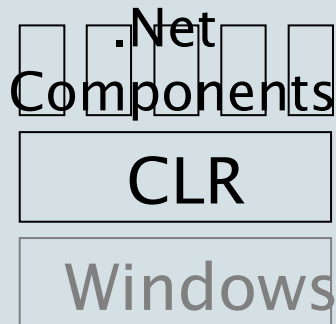
Use of Component Model



Component Platform

A component platform is run-time incarnation of the component model.

For example:



Component Platform

Platforms typically provide support for

Inter-component services

- binding
- interaction

Component lifecycles

- install, replace, remove

Extra-functional / resource aspects

- scheduling
- quality of service management
 - (dynamic) load balancing
 - (re)negotiation
- security
- fault tolerance (replication)
- interoperability (language/OS)

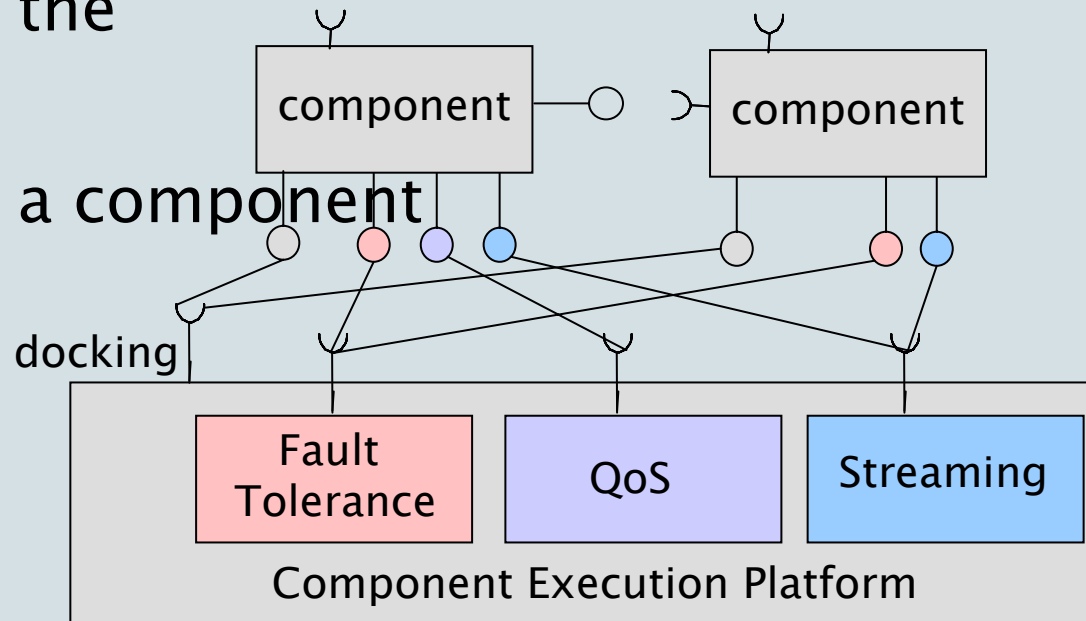
Interfaces are central to CBSE

- Interfaces describe what a component
 - .. may offer: **provides** interface
 - .. needs: **requires** interface
- Interfaces should be “first-class citizens”
Free combination of implementation and interfaces
- Components can have multiple interfaces,
corresponding to different types of access points

Multiple Interfaces

A component model may provide multiple (optional) frameworks

- 'Docking' a.k.a Binding
 - how to connect to the infrastructure
 - how to connect to a component
- Fault Tolerance
- Quality of Service
- Streaming
- Transactions ...



Characteristics of Components: Self-Describing

A component should be able to describe

- its functionality
 - public interfaces
- its behaviour
 - e.g. the events that it generates
- its variability/configurability
 - any properties that are customizable



component
specification

For run-time binding

Concluding Remarks 1

What's New in CBSE?

It deals with components that were not a-priori designed to be part of the same application

There will be multiple types of software components – belonging to multiple component models.

Concluding Remarks 2

CBD aims to

- increase productivity of SE
- change-ability of software systems

It is likely that every system will need some degree of custom development

- Wrapping is inevitable

CBSE is about managing dependencies

- Is this not addressing the symptoms, rather than the disease?
- Q: Why not try to avoid dependencies in the first place?

Type of dependencies:

design time: tooling

compile-time: compiler, other components

run-time: platform, other components

Scoping guidelines for components

A Component is a unit of abstraction

- Represents a single coherent concept/responsability.
- This is desirable for ease of understanding.
- This depends of the application level;
 - In an ERP system, a *bit*-component does not fit.

A Component is a unit of accounting

- In systems with multiple (classes of) users, cost is accounted per use.
- The overhead of accounting precludes small components.
- Components should correspond to units of

Example: word2pdf

A Component is a unit of analysis

- Unit of specification
- Types of analysis:
 - testing
 - type checking
 - version control

Example ...

A Component is a unit of compilation

- It should be possible to compile a component without using other components.
- Again, small components are not practical.
- This leaves open whether components should be compiled at all?
- Considering only executables as components avoids compile-time dependencies.

A Component is a unit of deployment

- Deployment is the process of readying a unit for operation in a particular environment.
- It carries all the information needed for preparing it for execution.

A Component is a unit of fault–containment

- Faults should be contained to a single component.
- Faults may not cause the component to display behaviour/produce results that is not specified by its invariants/postconditions.

Design alternatives:

- either include all exceptions in the interface
- or, ensure that no exceptions cross the component boundary

Special attention is needed for confining memory errors
(dangling pointers, memory leaks)

Language support is needed for intercepting errors

A Component is a unit of fault–containment

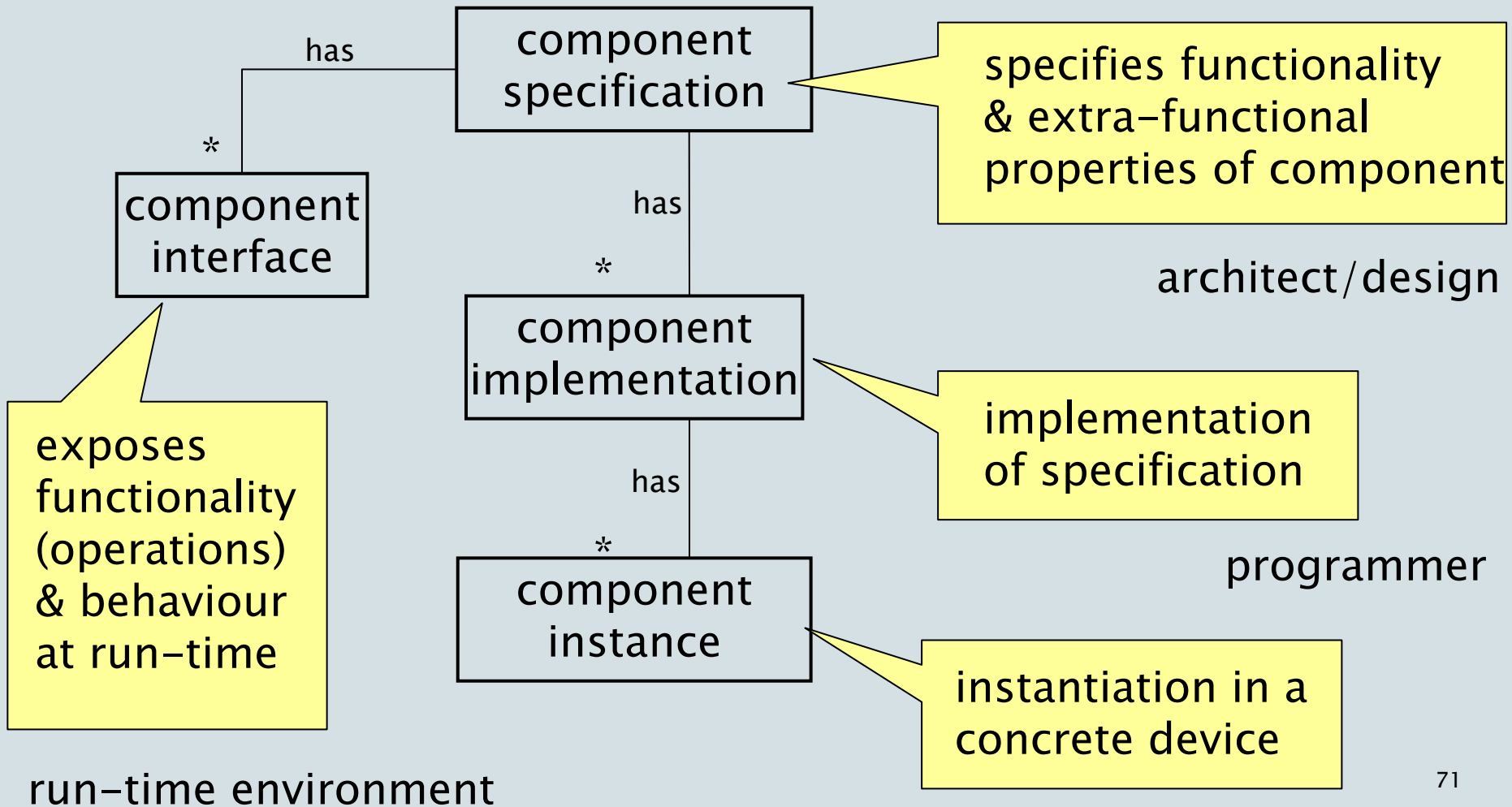
Proper handling of “unchecked exceptions” requires architectural attention.

→ Conventions must be part of the component model or of an application model.

For Example:

In distributed systems, no single component can be expected to solve the problems of node– or network–failure.

Useful Distinction



Object Technology and CBSE

“OO is Neither Necessary Nor Sufficient for CBSE”

OO was a useful and convenient starting point for CBSE

OO did not express full range of abstractions needed
by CBSE (*insufficiency*)

It is possible to realize CBSE without employing
OO(*non-necessity*)

On the relation between OO and CBD

- Object Orientation emphasizes modularity of the design of a system
- CBD emphasizes modularity in production, deployment and use of a system.