

Component-Based Software Engineering



M.R.V. Chaudron

Leiden Institute for Advanced Computer Science

–

Adapted by Tom Verhoeff for 2II45 in 2008, 2009

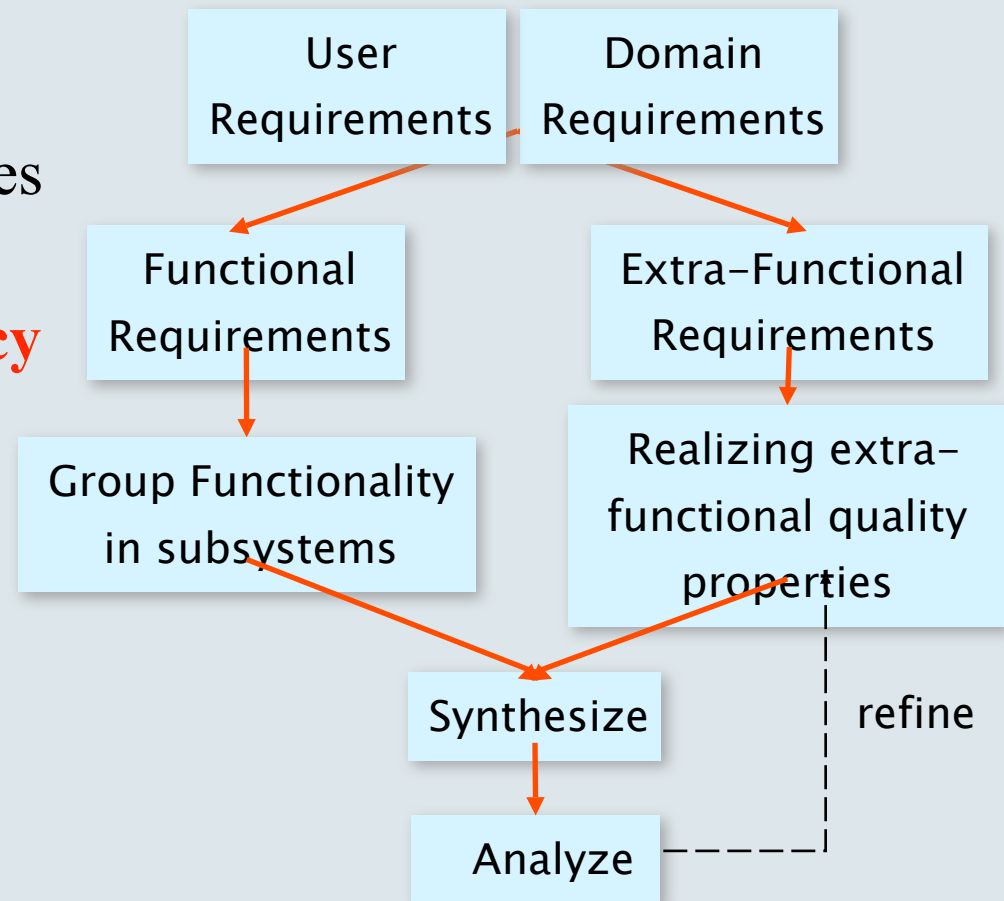
Contents

- ▶ introduction to Component-based SW development
 - ▶ motivation
 - ▶ main concepts
 - ▶ CBSE and reuse
 - ▶ component models, components
 - ▶ CBSE and object technology

Concerns for the Architect

The three major concerns for the architects are [Gacek95]

- 1) requirements **traceability**
why do we need this?
- 2) support of **trade-off** analyses
why do we chose this?
- 3) **completeness & consistency**
of the architecture
Do we have everything?



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, information hiding)
 - keep your options open
- Analyze at an early stage (use maths! and scenarios)
- **Simplify, simplify, simplify**
- Regularly update planning and risk analysis
- Monitor that architecture is implemented as intended
- Get good people, make them happy, set them loose

Recommendations for Architecture Description

- Describe the system **goals** & the **assumptions on the environment**
- Describe the design **principles, decisions, guidelines**
 - and their **rationale**
- Describe **several views** that can be combined in a consistent model at least the following views should be given:
 - **functional/structural (decomposition) view**
 - include a description of the interfaces between (sub)systems
 - **process/dynamical view**
 - **deployment view**
- Prevent mixing of views
- Address non-functional (*ilities) aspects
- Use a well-defined notation and include its **key/legend**
 - this aids systematic use of notation/avoids inconsistent use
 - improves common understanding
 - prevents mixing of different levels of abstraction
- Add explanation in **natural language**

If you haven't analyzed it, don't build it.

- Use **Scenario-based** evaluation for what-if questions
 - Identify **sensitivity- & trade-off points**, and **risks**
- Use **analytical** methods to support architectural decision making
 - Reliability → Reliability Block Diagrams
 - Throughput → Queuing networks
- Use **metrics** to manage
 - Modularity (coupling, cohesion)
 - conformance of implementation to design
- Many analyses are of 'back of the envelope' size.
 - little effort, lots of value
 - even if your model is not perfect (which they never are)

Component Based Software Engineering

References: Main Sources

Main text (background):

- Component Software: Beyond Object Oriented Programming
Clemens Szyperski, Addison-Wesley, (2nd ed, 2002)
- papers from course web page:
 - [Volume II: Technical Concepts of CBSE](#), F. Bachman et. al.,
CMU/SEI TR 2000-008, May 2000 [Must read]
 - [“Mass Produced Software Components.”](#) Douglas Mc Illroy,
1968/9, NATO Conference on SE [Optional]

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

Problems of Software Engineering

- The **size & complexity** of software increases rapidly
- Single products become part of **product families**
- Software is **upgraded after deployment**
- Applications must be built from components that were never intended to work together.
- The **time-to-market** must decrease significantly
- The **cost** of products must be reduced

Productivity & Flexibility

CBSE is part of the solution, but not in isolation

The CBD-‘Solution’

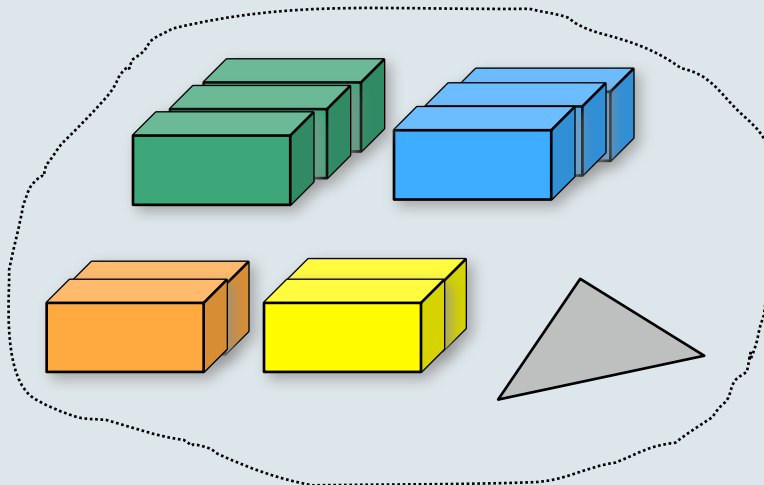
Systems should be assembled from existing components.

Idea dates (at least) to the 1968 NATO Conference.

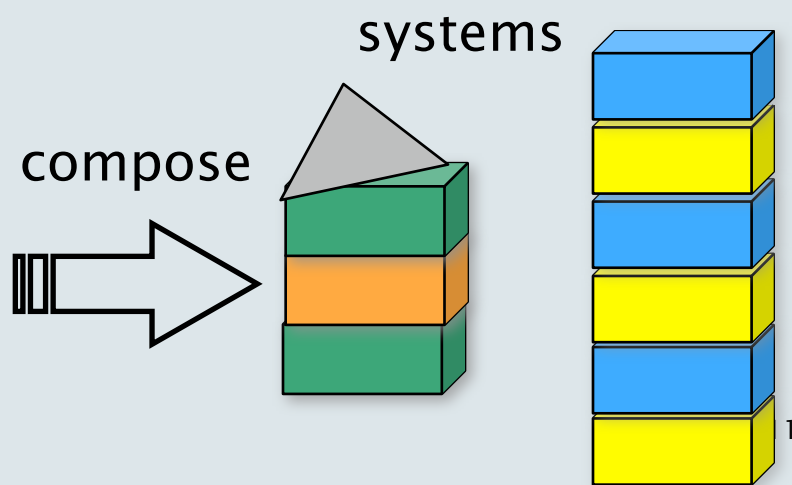
Douglas McIlroy: **Mass Produced Software Components**



component repository & market



component-based



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?

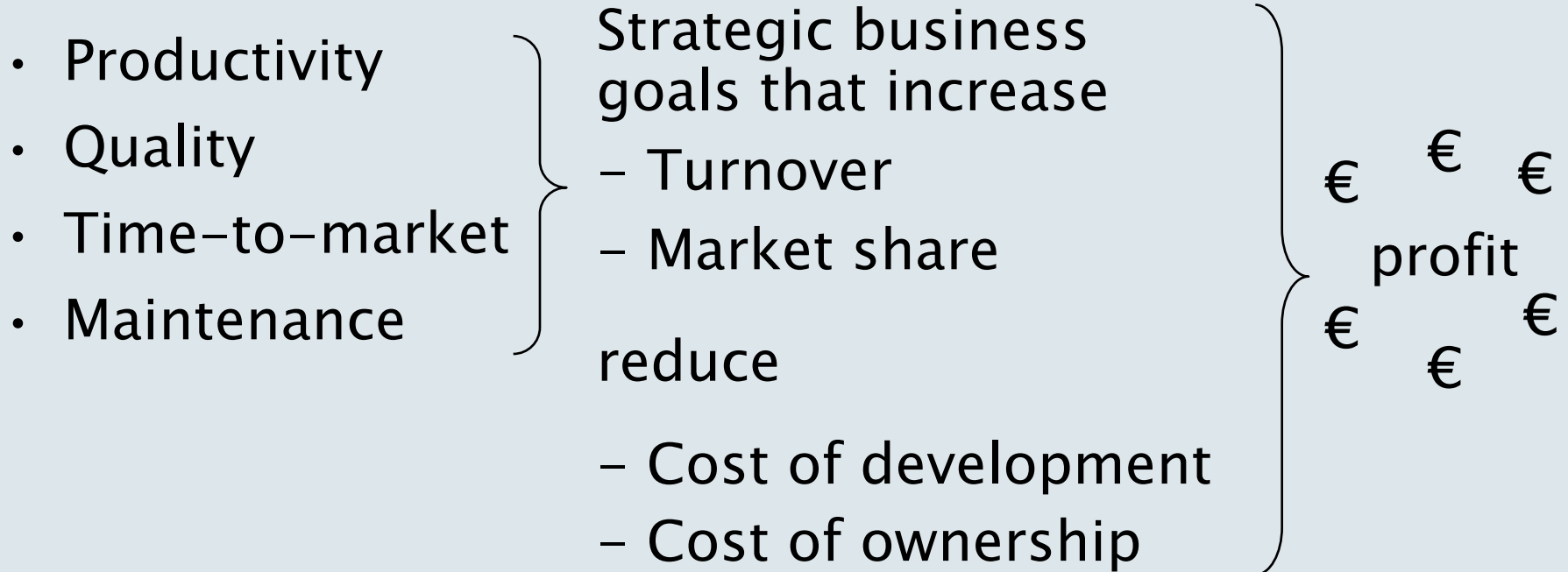
based on definition of SEI in CMU/SEI-2000-TR-008

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

Motivations for CBSE



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 uses of component **improves** its **quality**, increases likelihood of finding&removing defects.

CBSE & Time-to-market

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

CBSE & Maintenance

The use of CBD requires good modular design.

This modularity provides quality properties like

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

Technical Drivers for CBSE



CBSE may help improve system qualities

What is software reuse?

Software reuse is the process whereby an organisation employs a set of **systematic operating procedures** to specify, produce, classify, retrieve, and adapt software artifacts for the purpose of using them in its development activities.

Mili et.al. 2002

Reuse-based Software Engineering

- Reuse-based SE has many business drivers in common with CBSE:
 - increase productivity & quality
 - reduce time-to-market,

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

CBSE enables Reuse, Reuse is not sufficient for CBSE.

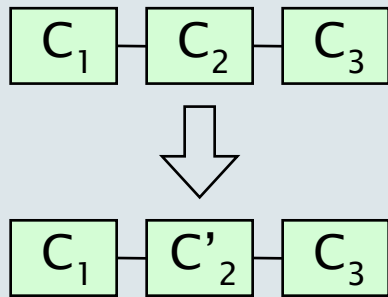
Reusable Assets

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

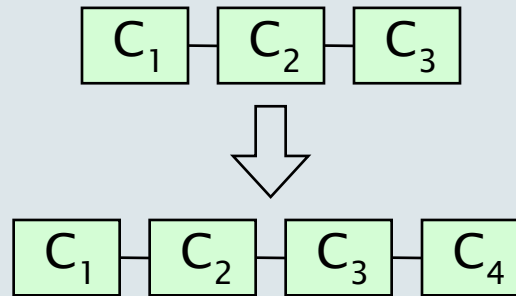
Distinguishable flexibility requirements

Substitutability



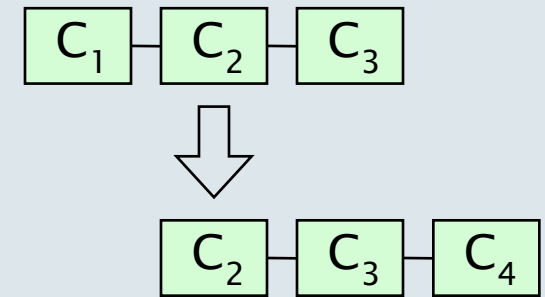
complete
specification

Extensibility



extensible
architecture

Decomposability

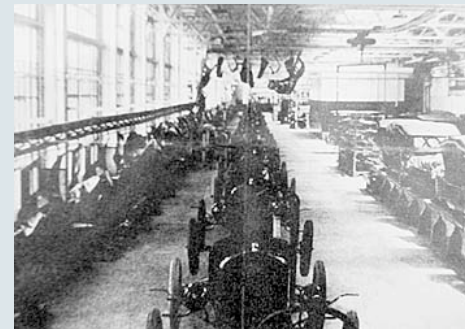


generic components,
flexible architecture

At what stage is flexibility needed?
Design-, compile-, run-time

Modularity in different phases

- **Modular in Design**
 - Modern computers
 - Eclectic Furniture (not “modular” furniture)
 - Recipes in a cookbook
- **Modular in Production**
 - Engines and Chassis
 - Hardware and software
 - NOT chips, NOT a cookbook
- **Modular in Use**
 - “Modular” furniture, bedding
 - Suits and ties
 - Recipes in a cookbook



COMPONENT MODELS

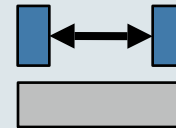
Michel Chaudron

Interfaces are central to CBSE

Standardization to avoid market fragmentation

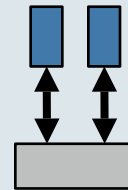
- Horizontal standards:

Internet, OS's, UI's, ...



- Vertical standards:

Control, ERP, TV, ...



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

What is a Component?

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

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

A component is a piece of software small enough to create and maintain, big enough to deploy and support, and with standard interfaces for interoperability.

Jed Harris, President of CI Labs (Jan. 1995).

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

Cross-cutting concerns

What is a 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 independent deployment?

A software component is a unit of **independent deployment**

“It works fine on my machine.”

- never partially deployed
- no dependencies on peer-components
 - some ‘meaningful’ functionality by itself
 - components tend to be ‘large grained’
- Components should be **deployment-friendly**

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 provide some design guidance.

What is cohesive? loosely coupled? single abstraction?

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

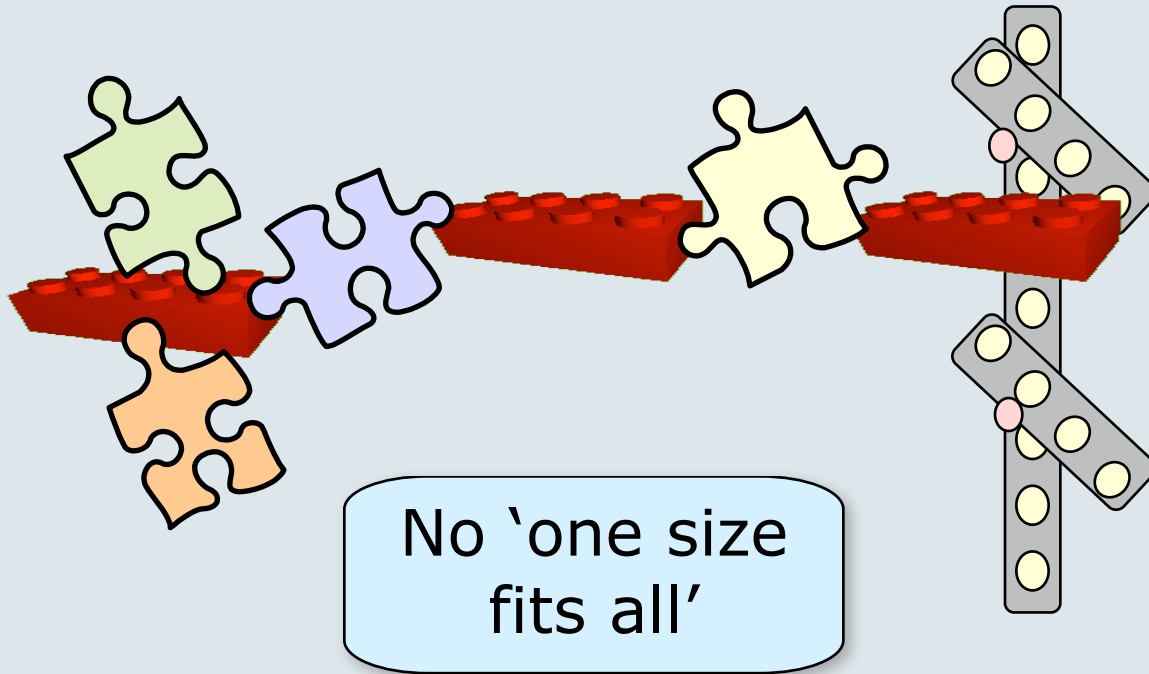
“A replaceable unit of development work which encapsulates design decisions and which will be composed with other components as part of a larger unit.”

– Desmond D’ Souza, in Catalysis

What is composition?

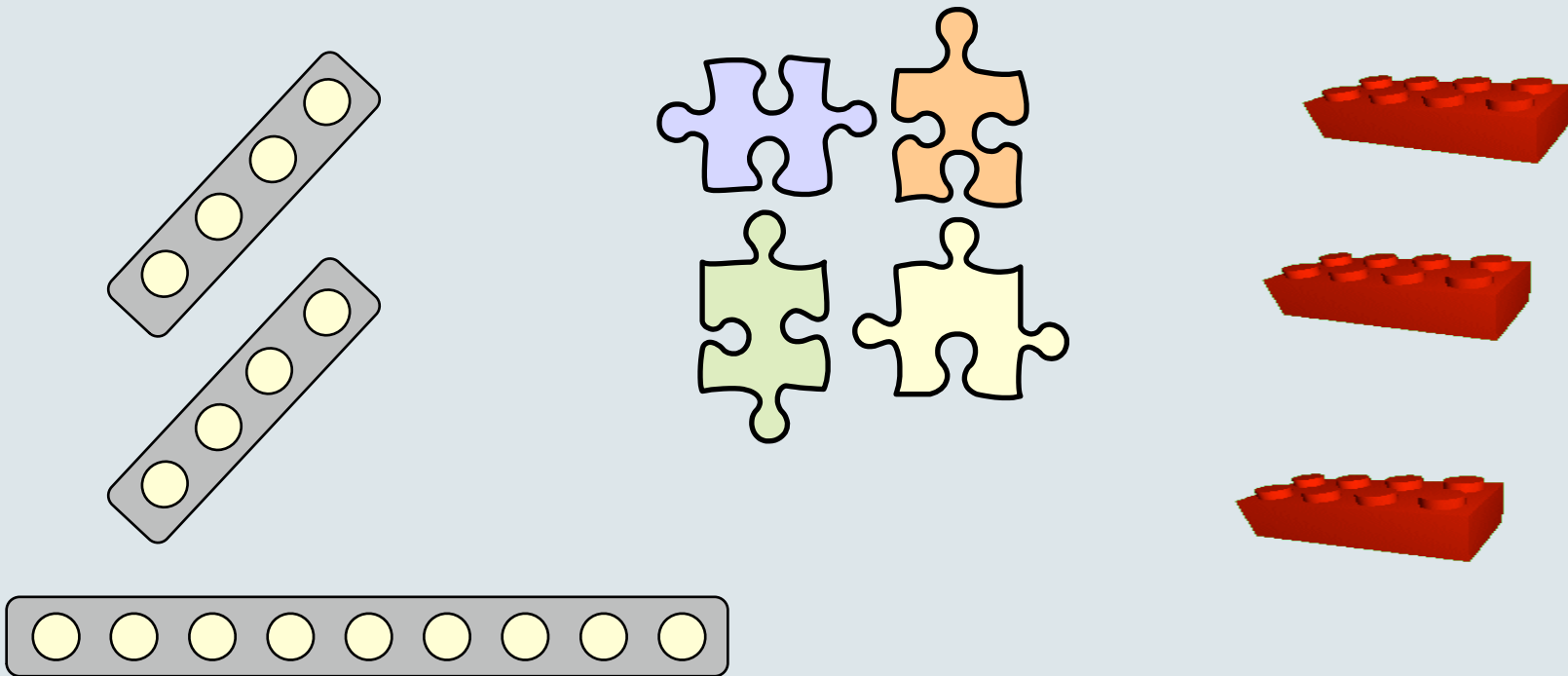
- What is composition?
- What things can hinder successful composition?
- How does composition depend on the ‘matter’ of a component?
- May require specific **infrastructure**

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 + ...

Object Technology and CBSE

“OT is Neither Necessary Nor Sufficient for CBSE”

OT was a useful and convenient starting point for CBSE

OT did not express full range of abstractions needed by CBSE

(insufficiency)

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

CBSE might induce substantial changes in approach to system design,
project management, and organizational style

On the relation between OO and CBD

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

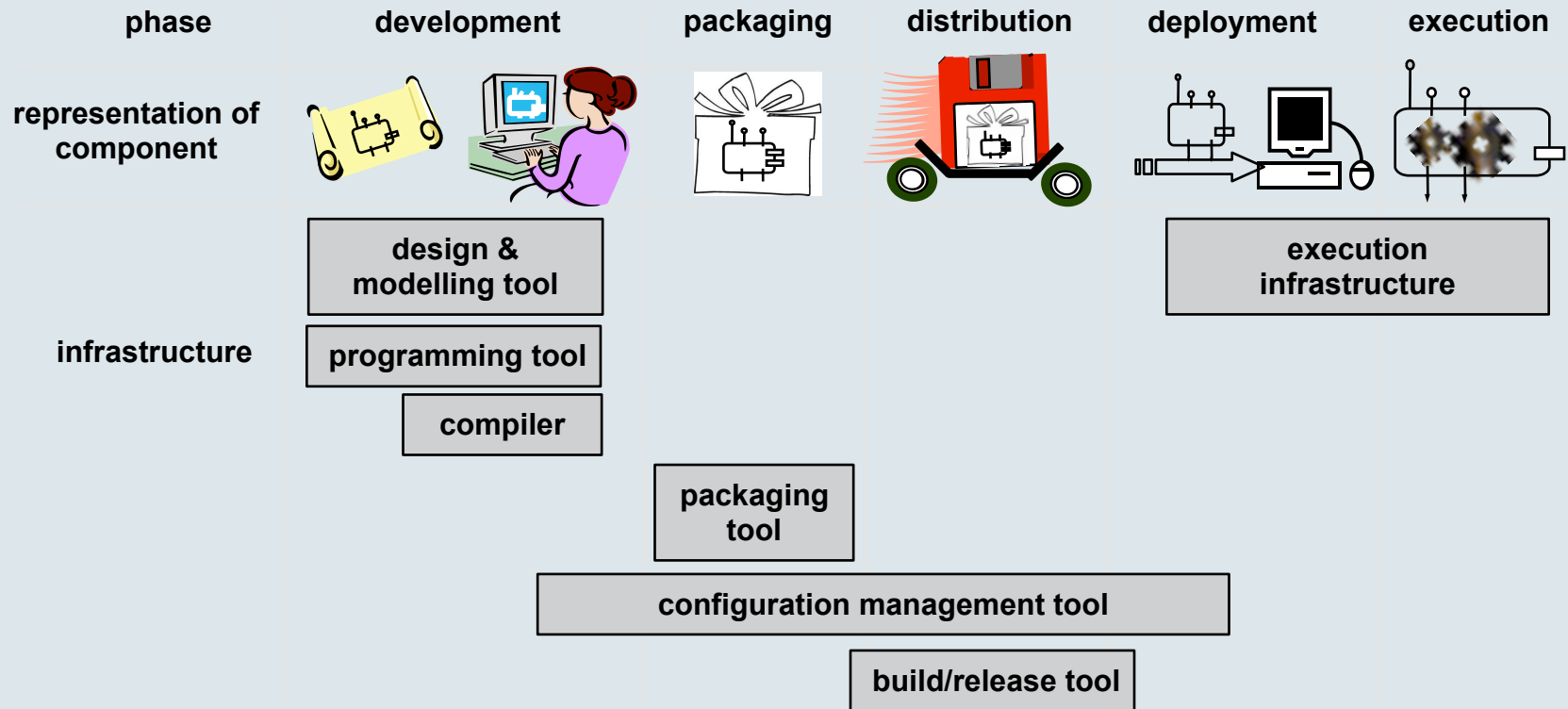
Objects vs. Components

- “Object Oriented Programming = Polymorphism + (Some) Late Binding + (Some) Encapsulation + Inheritance
- Component Oriented Programming = Polymorphism + (Really) Late Binding + (Real, Enforced) Encapsulation + Interface Inheritance + Binary Reuse”

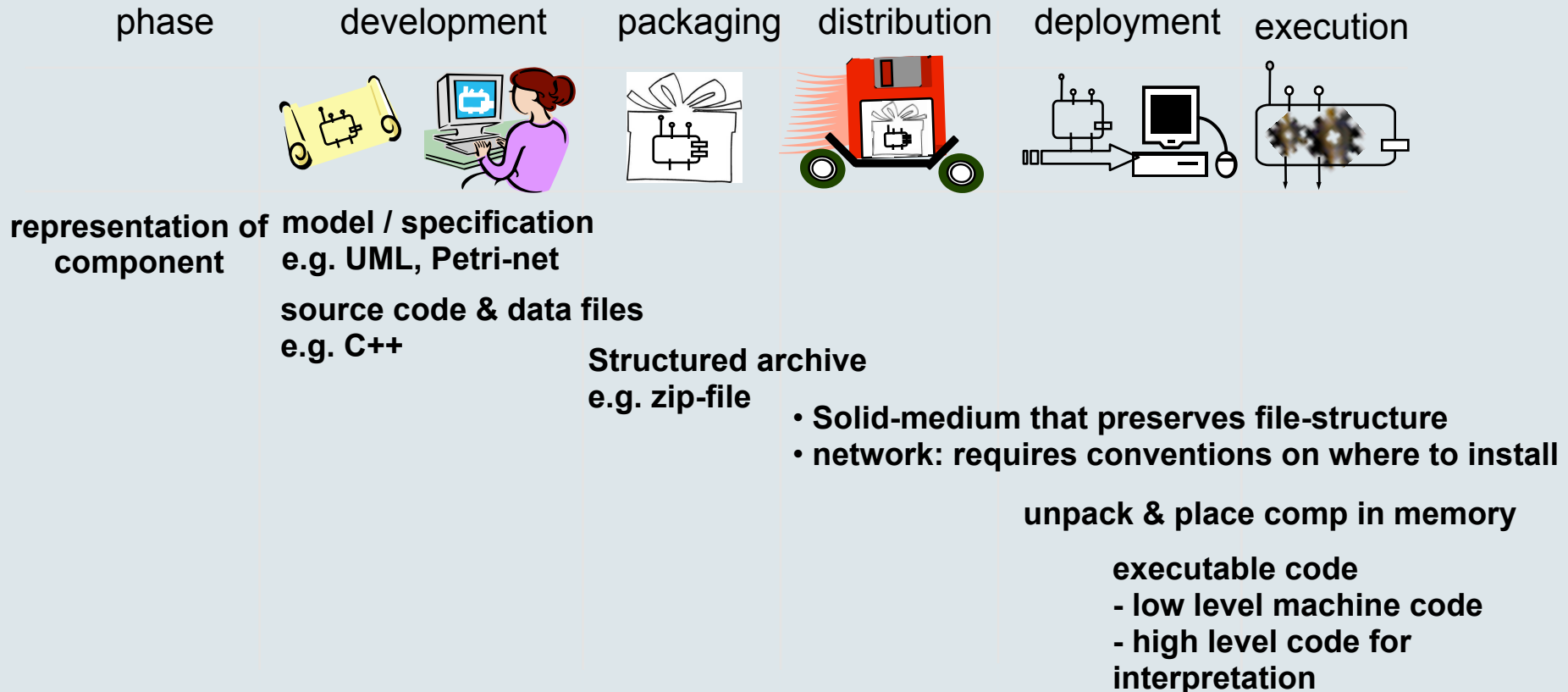
Charlie Kindel “COM Guy” Microsoft Corp. 9/97

Questions of CBD approaches

- 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?
- What happens if one component needs to be shut down temporarily? Can this be hidden from (made transparent to) other components?
- How does development and evolution of one component of a family of systems impact other components?
- Are components bloated by code that is unrelated to its specific task in a system?
- 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?
- How can components that run on different platforms interoperate?



Component Development Lifecycle



In different processes, components are composed in different phases of development

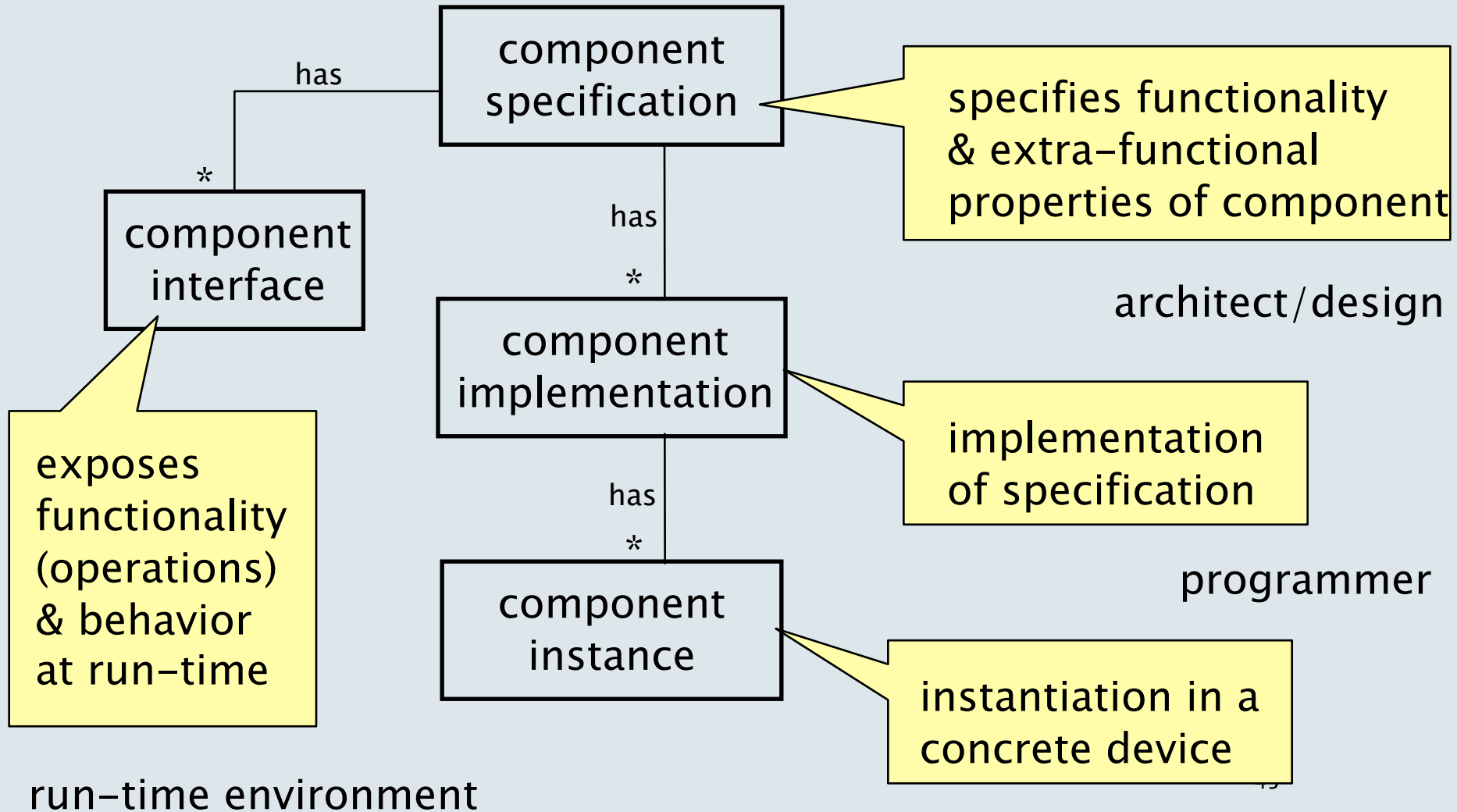
Component Model

What conventions does a component model need to specify to enable composition?

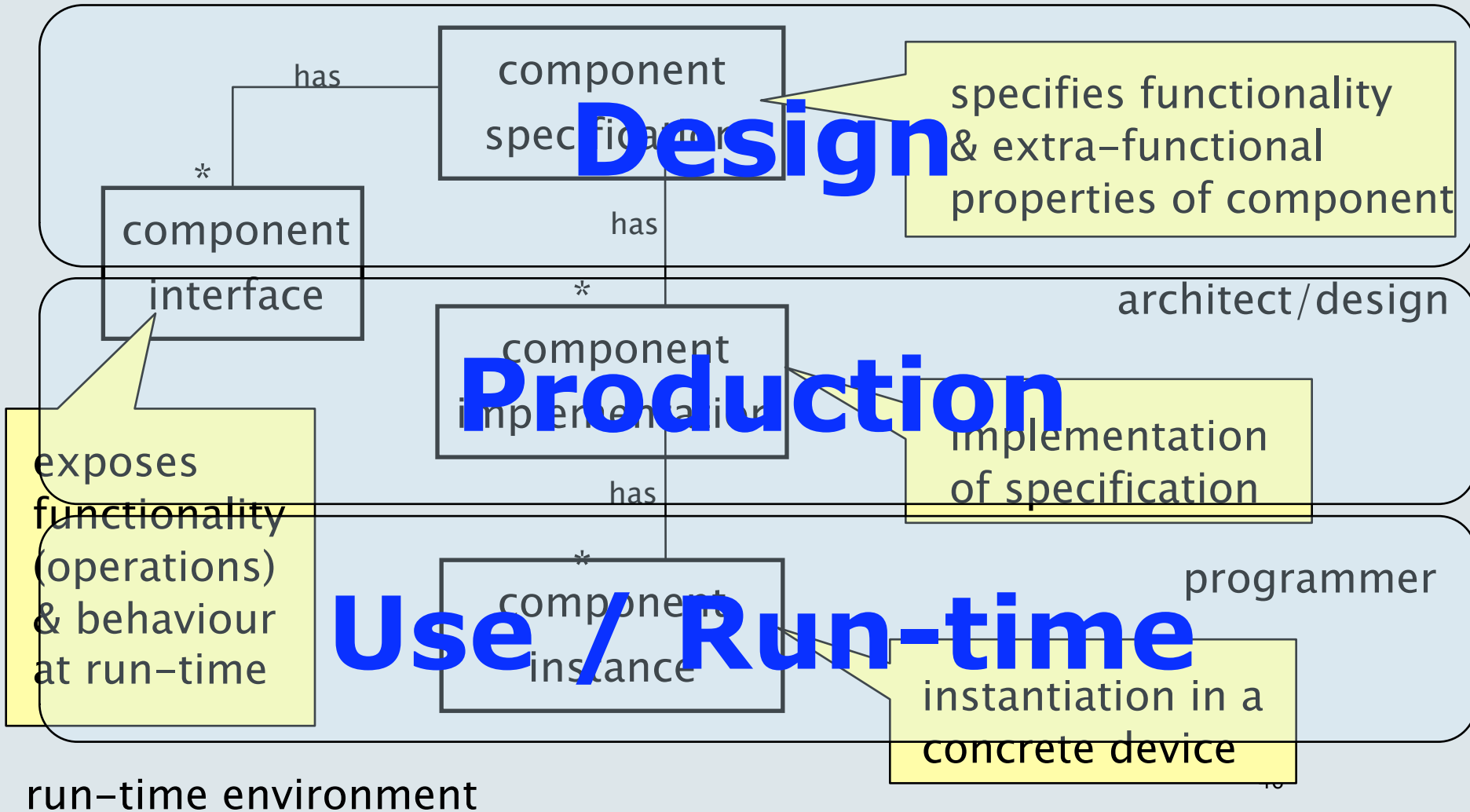
Standards for

- Implementation technology
- Standard interfaces
- Specification/documentation/meta-data

Useful Distinction



Useful Distinction



Component Model

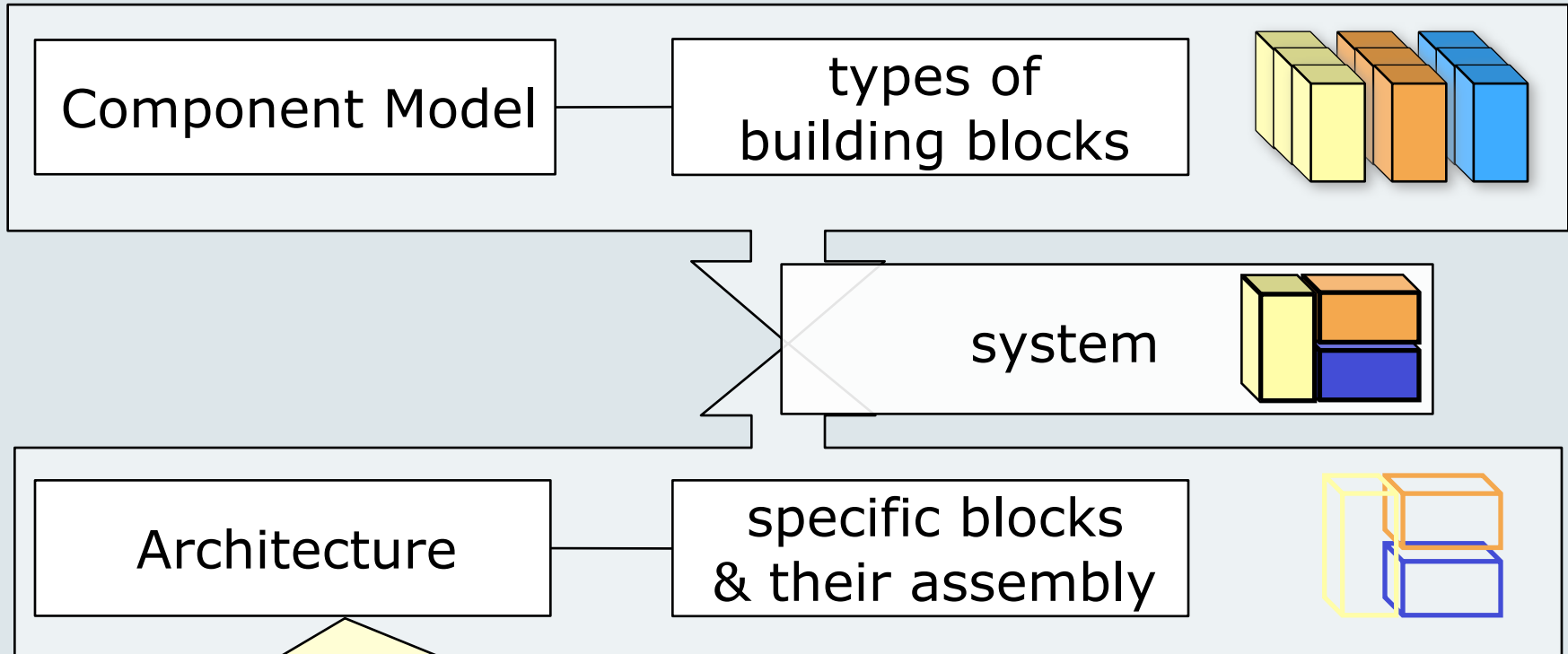
Definition: A **component model** specifies the standards and conventions that are needed to enable the composition of independently developed components.

Typically:

- The composition mechanism/infrastructure
- The conventions that components must adhere to in order to enable successful composition

Definition: A **component** is a building block that conforms to a component model.

Architecture & Component Model



defines

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

Questions?

You should know

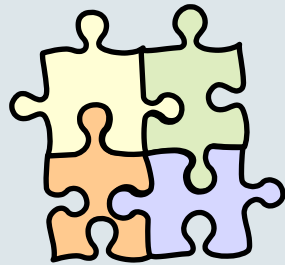
- an answer to ‘What is a component?’
- what a component model is
- the relation between reuse, CBD, and OO
- the relation to architecture

Self-Study material:

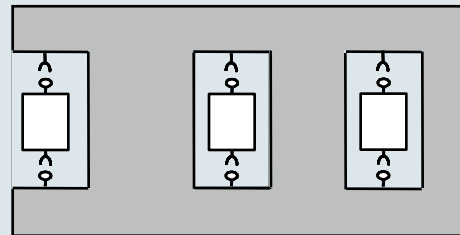
- Tech. Report SEI: [Technical Concepts of CBSE](#)
- Optional paper by McIlroy: Mass Produced Software Component

Architecture vs. Generic Components

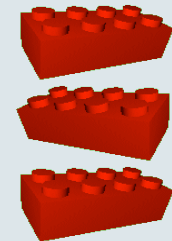
Architectural component



Framework component 'plug-in'



Generic component



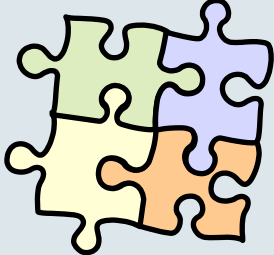
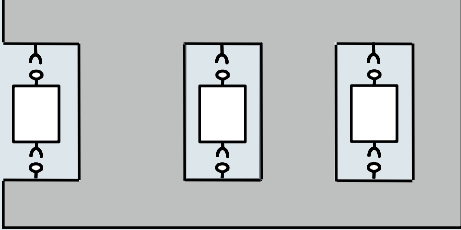
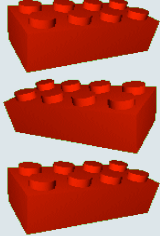
there is exactly one component that fits each place in the system

several components fit at a particular place in the system

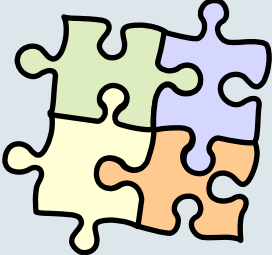
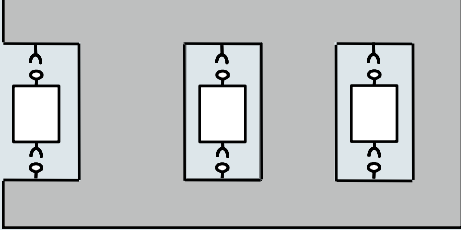
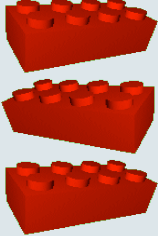
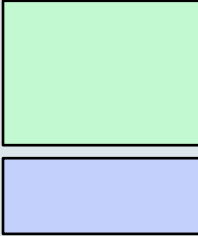
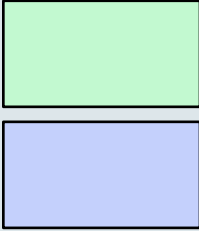

every component fits at every place



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		
<p>architecture driven features</p> <p>basic interoperability</p> 		

A Component-based Reference Architecture for Computer Games (E. Folmer, 2007)

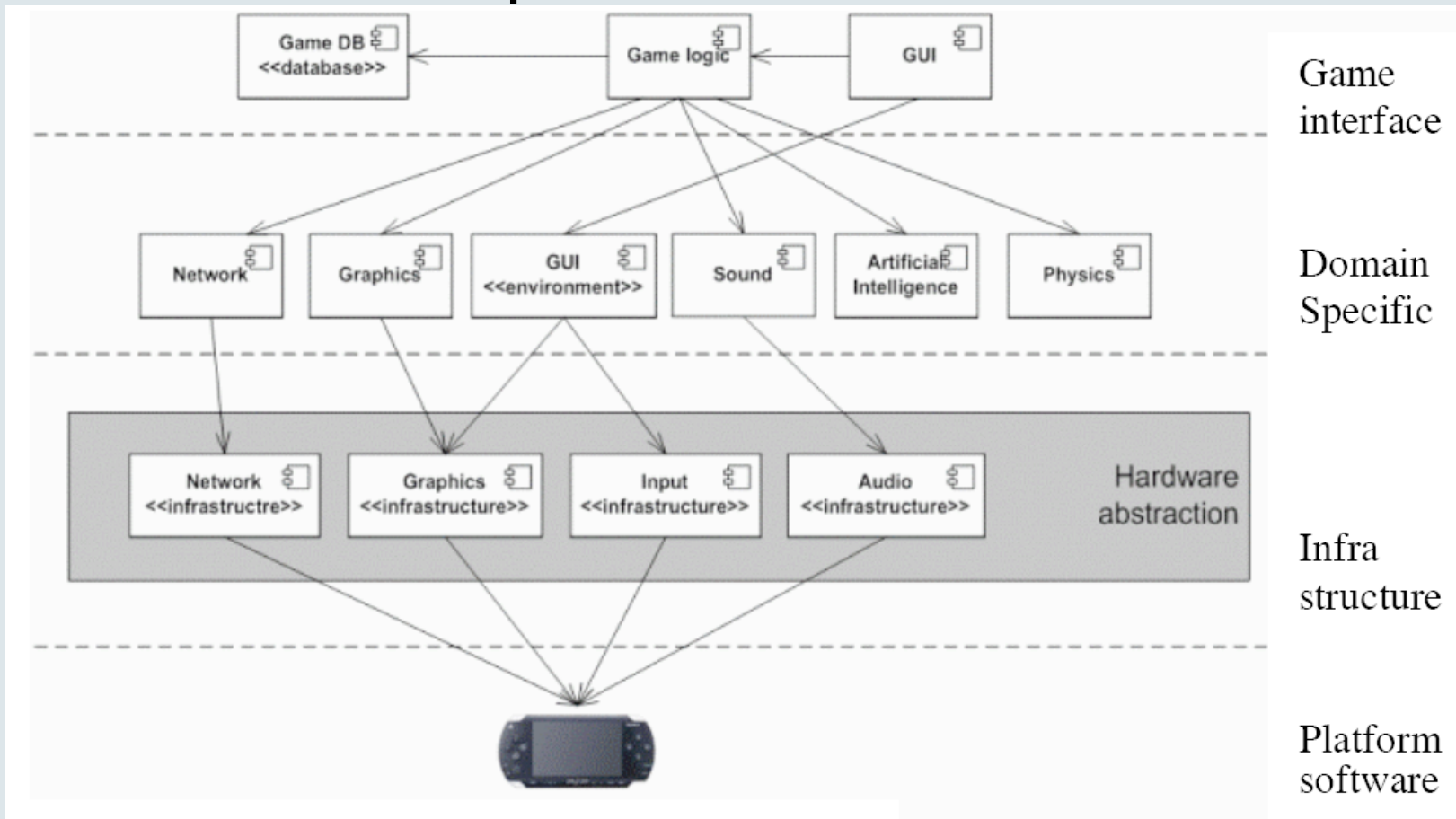
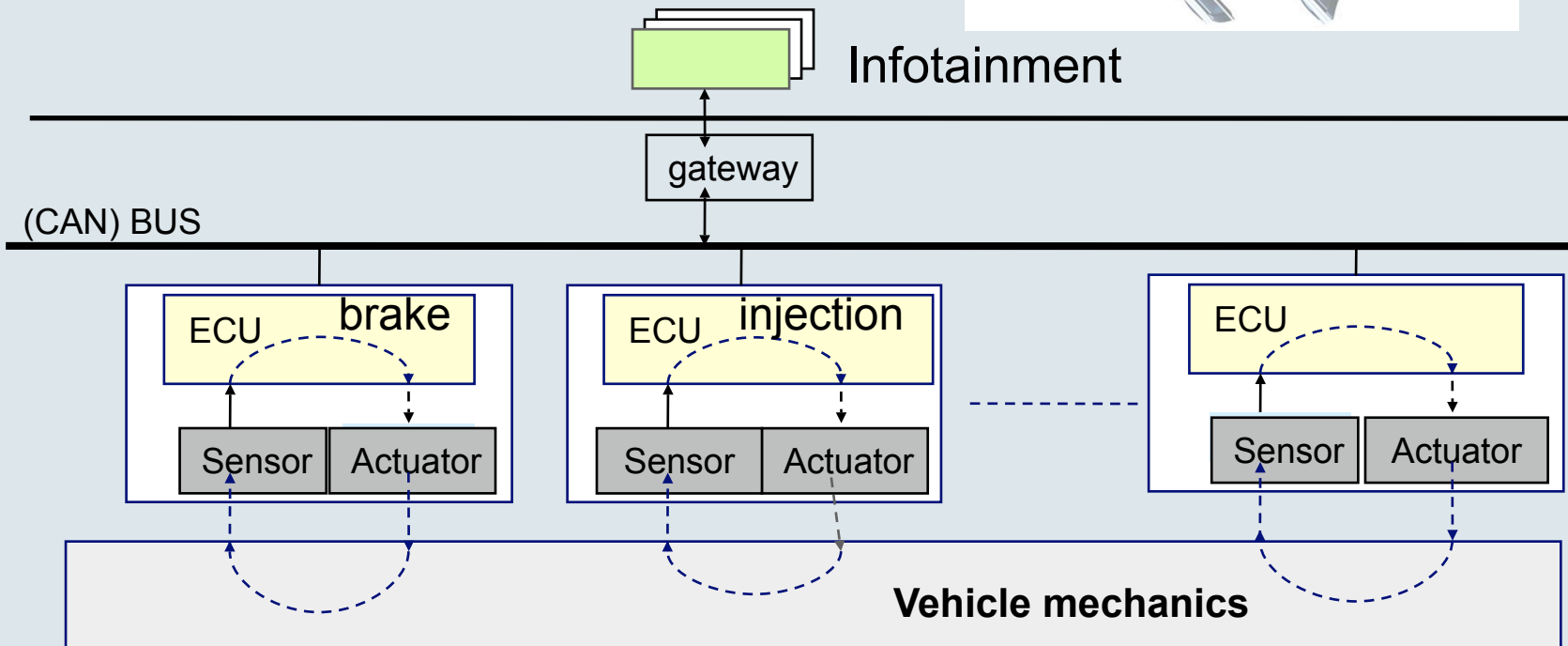
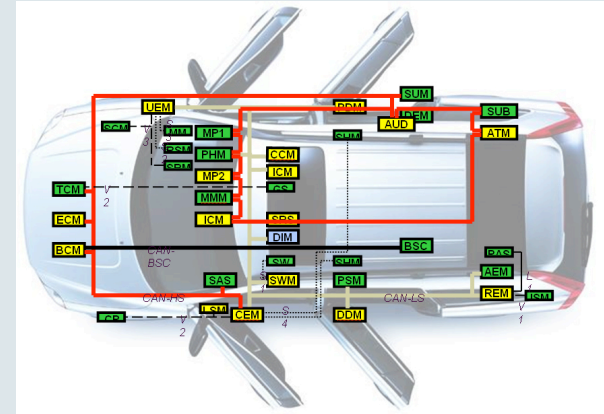


Fig. 1. A reference architecture for the games domain

CBSE in Automotive



ECU – Electronic Control Unit

Software Component Technology

Definition A Software Component Technology is the implementation of a component model by means of:

- standards and guidelines for the implementation and execution of software components
- software tools that supports the implementation, assembly and execution of components.

Examples: (D)COM, .Net, Corba-Component Model, Enterprise-Java Beans, Koala, Robocop, Fractal, ...

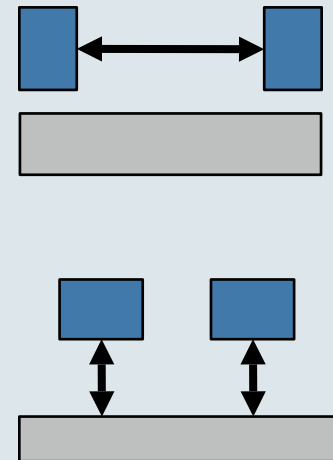
Aspects of Component Models

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

A component model typically addresses

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

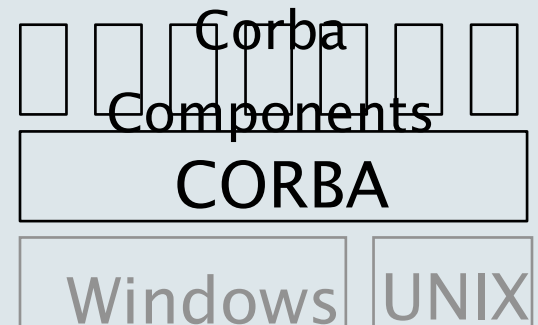
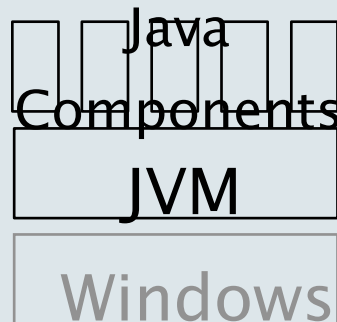
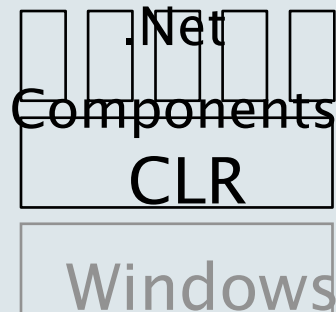
Related: Packaging Model



Component Platform

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

For example



Component Platform

Component Platforms may provide support for

Inter-component services

- binding
- interaction

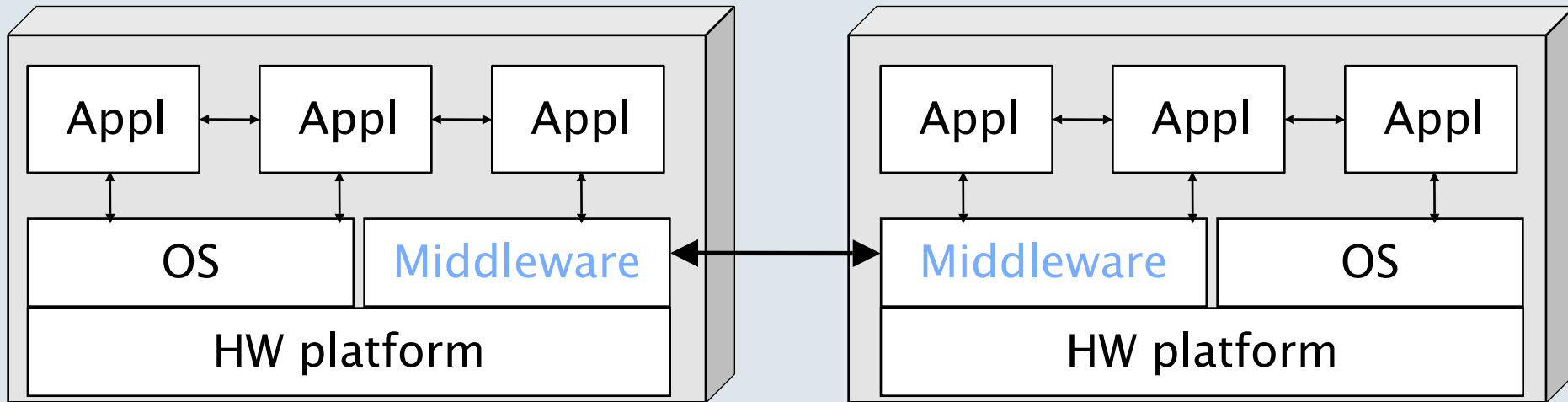
Component lifecycles

- install, create, replace,
start, stop, remove

Extra-functional / resource aspects

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

Middleware in Distributed Systems



Middleware is software that connects two or more software applications so that they can exchange data.

"the software layer that lies between the operating system and the applications on each side" of a distributed computing system.

Concluding Remarks

- A component is a building block that conforms to a component model
- Different component models aim to achieve different system quality properties
- A component model may be biased to specific types of architectures