Karlsruhe Institute of Technology

Software Engineering II

Prof. Dr. Ralf H. Reussner

Topic 05

Software Components

DSIS – DEPENDABILITY OF SOFTWARE-INTENSIVE SYSTEMS
KASTEL – INSTITUTE OF INFORMATION SECURITY AND DEPENDABILITY

dsis.kastel.kit.edu

Overview on Next Lectures

Content

- introduction to software components
- overview of technical component models
  - and web services
- scientific component models
  - Palladio

Learning Goals: Be able to

- describe what component models define
- describe the elements of the Palladio component model (PCM)
- explain some of the design decisions made in the PCM

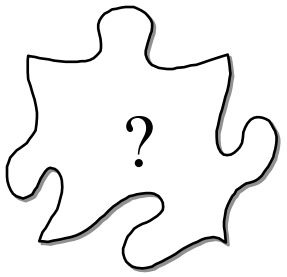
SOFTWARE COMPONENTS

Software Components

- Building blocks for software
  - a natural concept, however, somewhat difficult to define
- Also, not a new idea!
  - first proposed by McIlroy (1968 at the NATO-Conference in Garmisch)
  - idea was to allow software reuse

- *"Components are for composition, much beyond is unclear…"* (Szyperski)

- Frequently asked questions
  - does a component have state?
  - is a component (more or less than) an object?
  - is a component a module?



Definition Attempts

- The term component is often just used in the generic sense of building block
- However, various authors have attempted to give a more precise definition

> *a component represents a modular, deployable, and replaceable **part of a system** that encapsulates implementation and exposes a set of interfaces "*
>
> – UML 1.5 Specification

> *a reusable software component is a logically cohesive, **loosely coupled module** that denotes a single abstraction "*
>
> – Booch 87

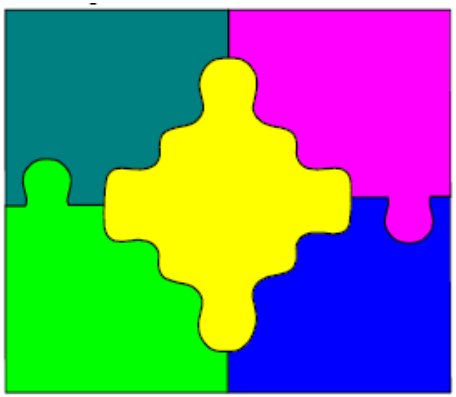
- Today's most commonly accepted definition

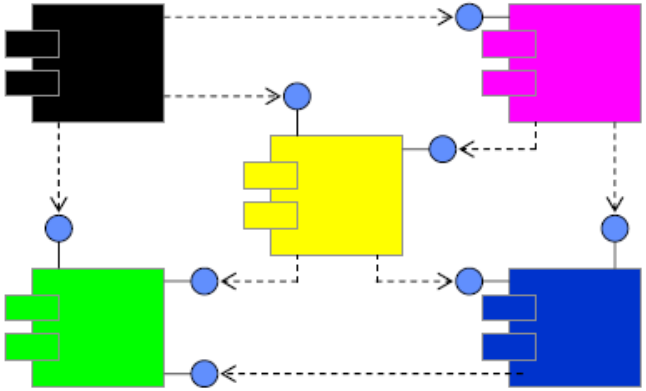
> *A software component is a **unit of composition** with **contractually specified interfaces and explicit context dependencies** only.*
> *A software component can be deployed independently and is subject to composition by third parties."*
>
> – WCOP'96 [Szyperski]

This is a Component

- *"A component is a contractually specified* building block *for software which can be* composed*,* deployed*, and* adapted *without understanding its internals."* – [Reussner2016]
  - not necessarily black-box: information on component's internals may be provided for tools
  - Minimize effort for composition, deployment, or adaptation

➔ Systems can be composed from components





Why Object CANNOT equal Component

➔ Inheritance is conflicting with the Black-Box-Principle of Components

```
class A {
   public T m {
      ... x(); ...
   }
   public T x(){...}
}
```

```
class B extends A {
   public T x(){
        ... // redefinition
   }
}
```

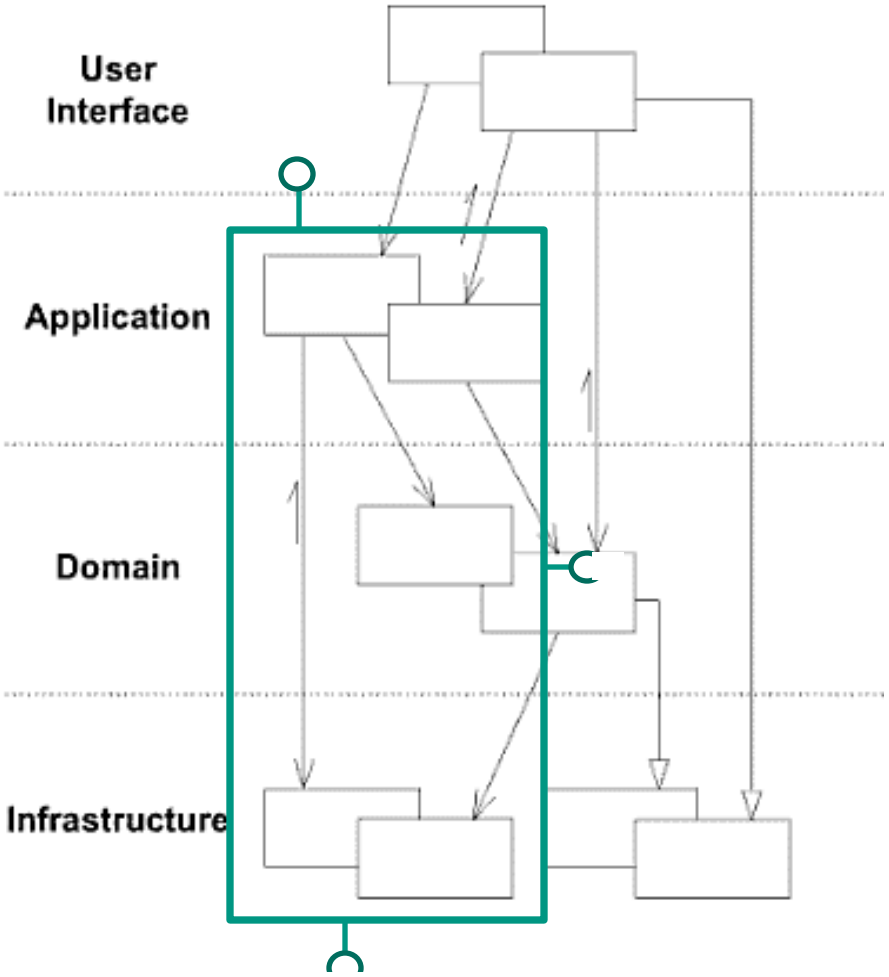
```
...
B b = new B();

b.m(); // result unclear
       // for developer of B and A!
```

Components in Layered Architectures

- **Components** are feature-oriented

➔ comprise the application and domain layer
  - sometimes called business components in an enterprise context

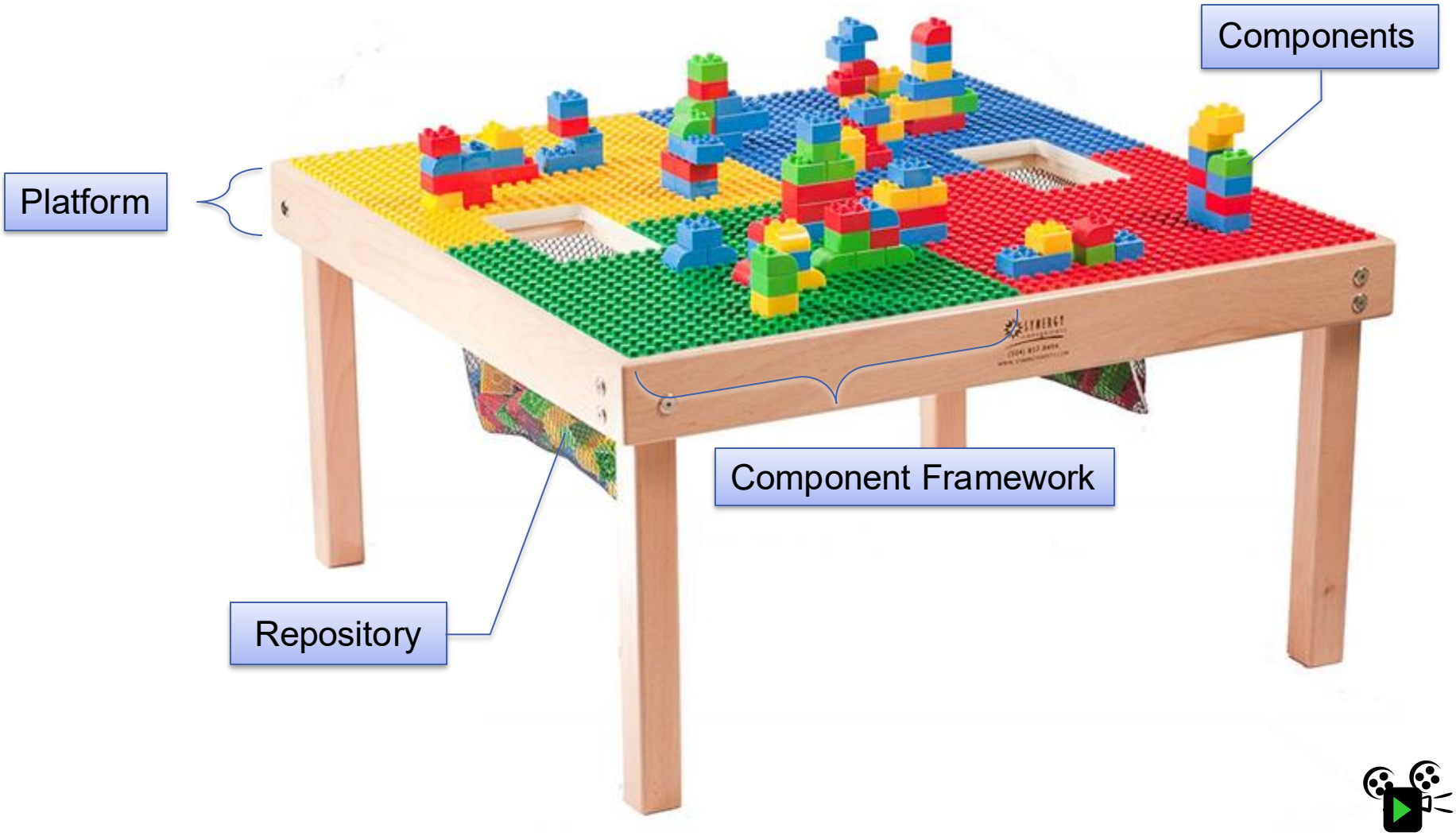
- infrastructure may also form nice components
  - for example JDBC



Component Model

- Concrete realisation of the principles of CBSE
  - in practice, only components built for the same model can interoperate
- Defines –
  - What a component is?
  - How does a component offer services?
  - How are components connected / composed?
  - How do components communicate?
  - Where can components be found?
- Often also providing an infrastructure for executing components
  - sometimes denoted component framework

Component Framework Visualization



Components in Java

- There is no such thing as a *component in Java*
  - ➔ how can we mimic them there?

- Packages
- Classes with dependency injection (for requires interfaces)
- Classes with inner classes (and dependency injection)
- Façade
- …

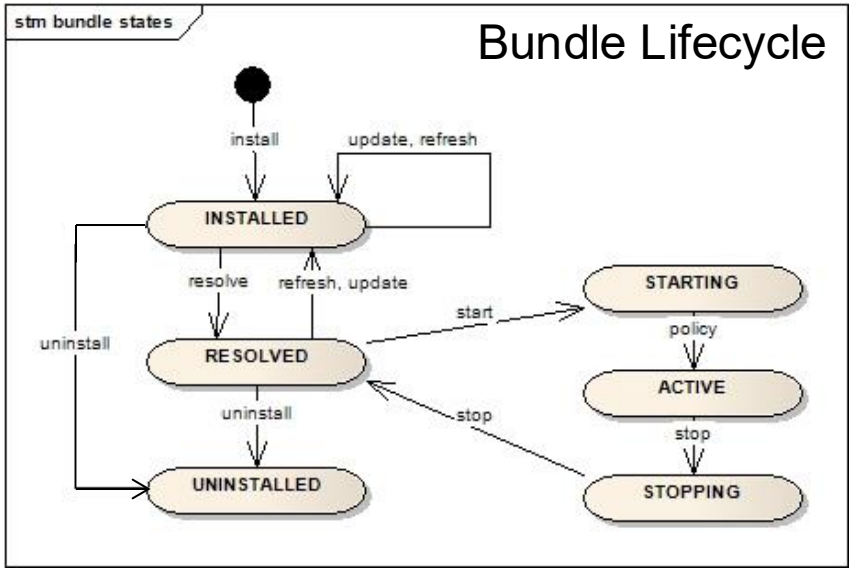
Technical Realisation of Components

- Technical realisation of CBSE ideas in recent years
  - OMG way: CORBA, CCM, OMA
  - Sun way: Java, JavaBeans, EJB
  - Microsoft way: COM, OLE/ActiveX, COM+, .NET CLR
- Similarities
  - late binding, encapsulation, interface inheritance, ...
- Differences
  - memory management, evolution and versioning, and many more
- Main Problem
  - most approaches are rather object-oriented than component-based
    - **CORBA** = Common Object Request Brokerage Architecture
    - **EJB** = Enterprise Java Beans (POJOs today)
  - no proper explicit required interfaces

➔ *Is OSGi the solution?*

OSGi *(Open Service Gateway Initiative)*

- OO has driven modularisation to the extreme
  - many fine-grained objects
  - but no real module concept „above“ packages
- OSGi offers bundles as a solution
  - basically JAR files defining a public interface via a manifest
  - *provided* services of bundles only usable when explicitly required in *consumer* bundle
  - integrated into Eclipse
- OSGi provides a registry for services (bundles)
  - only existing during runtime
  - services can come and go



https://jaxenter.de/erste-schritte-mit-osgi-9019

What Are Web Services?

*"Web services […] are **self-contained**, **self-describing**, modular applications that can be **published**, **located**, and **invoked** across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes. […]*

Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service."

– IBM

- **Self-Contained**

  functionality and attributes are exposed in a public interface while implementation is hidden

- **Self-Describing**

  have a machine-readable description that can be used to understand their interface

- **Modular**

  are reusable and can be composed to generate higher level functionality

- **Published**

  can be registered in electronic "yellow pages" for easy location by other applications

- **Located**

  are tied to a fixed, globally unique location identified through a URI

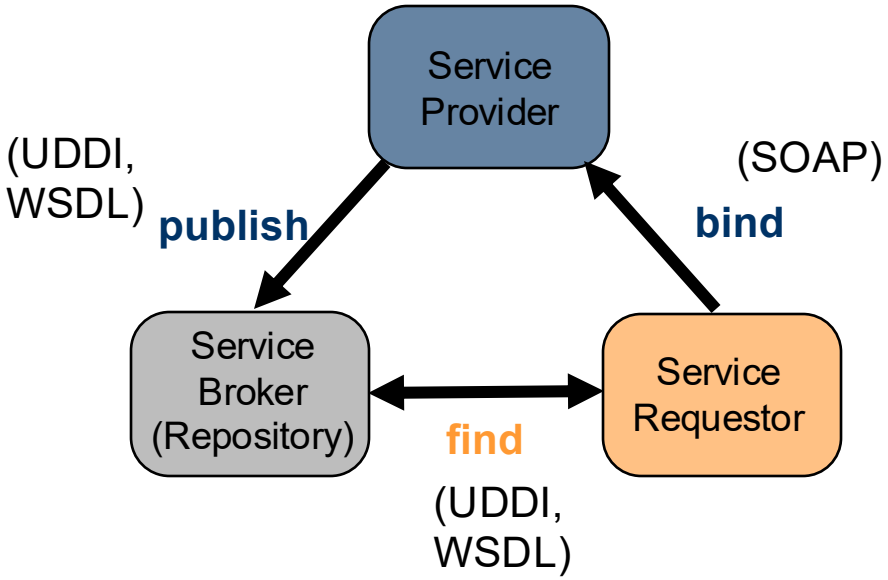
- **Invoked**

  can be invoked using a standard Internet protocol

➔ ***(Web) services can be seen as deployed components [Reussner]***

Service-Oriented Architecture (SOA)

- Elements in a system built from web services play one of three roles
  - service requestor
  - service provider
  - service broker (repository)



- service providers **publish** services by advertising service descriptions in service registry
- service requestors use **find** operation to retrieve service descriptions from service registry
- service requestors **bind** to service providers using binding information found in service descriptions to locate and invoke service

Core Web Service Technologies

SOAP (t.a.f.k.a. *Simple Object Access Protocol*)

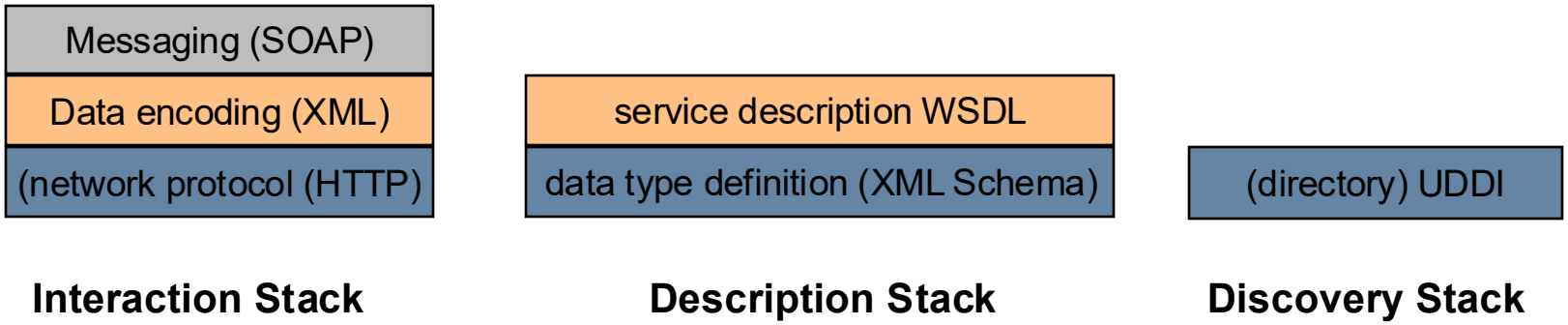
- message layout specification defining a uniform way of passing XML-encoded data
- simulates RPCs over standard Web communication protocols

WSDL (Web Service Description Language)

- defines Web Services as collections of network endpoints or *ports*
- port defined by associating *network address* with a *binding*

UDDI (Universal Description, Discovery and Integration)

- provides mechanism for clients to find web services
- basis for repository services for business applications

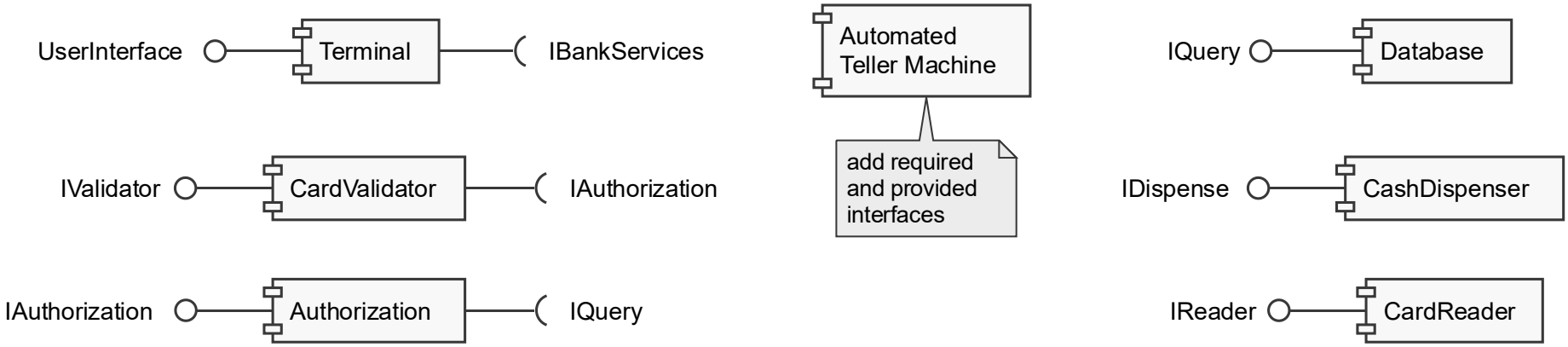


Research-based Component Models

- Aim for a stricter adherence to CBSE principles
- SOFA
  - protocol checking
  - compositionality
  - provides own infrastructure
- ROBOCOP
  - consumer electronics (e.g. TVs)
  - analysis of non-functional properties
  - provides own infrastructure
- KobrA
  - UML-based, hierarchical modelling approach
  - support for software product lines
- Palladio
  - performance prediction at design time
  - support of CBSE role model
  - mapping to EJB possible

Component Jigsaw Puzzle

- Design component model of automated teller machine (ATM)
- Use all shown third-party components
- Add *required* and *provided* interfaces to **ATM component**



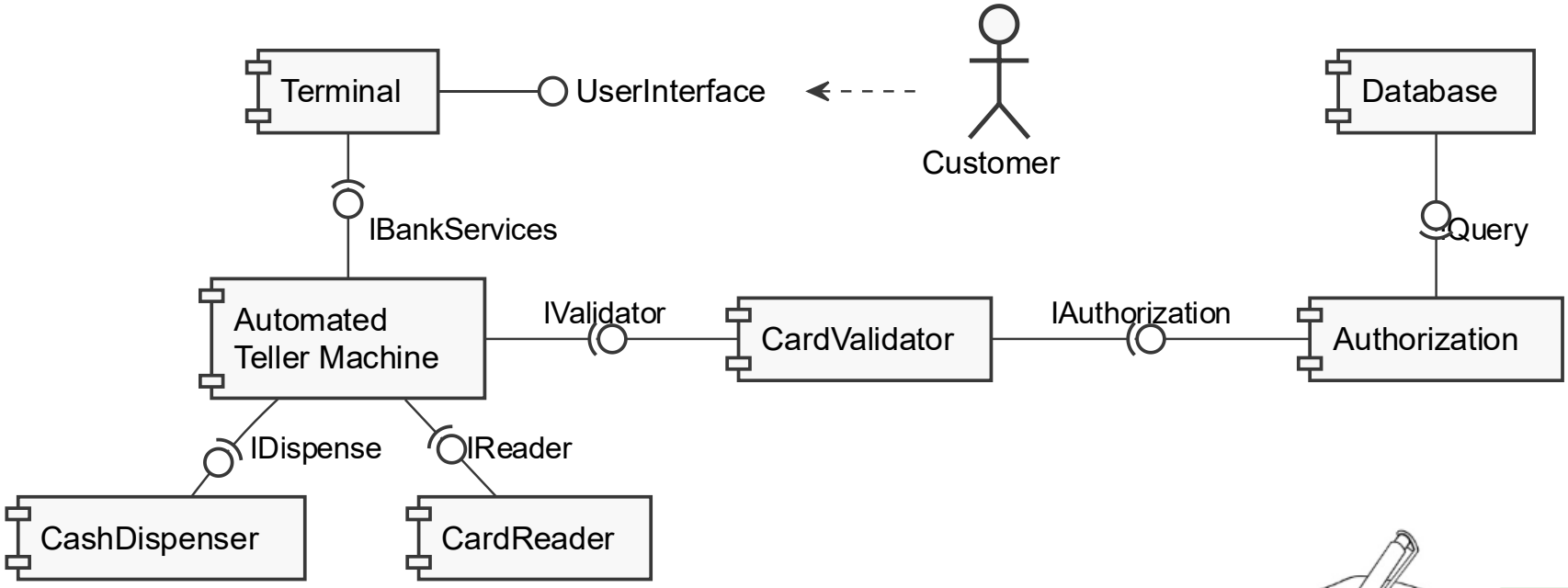
Which interfaces should the ATM *provide* and *require*?

Which components are directly connect to the ATM?

Draw the corresponding Component Model?

Component Jigsaw Puzzle

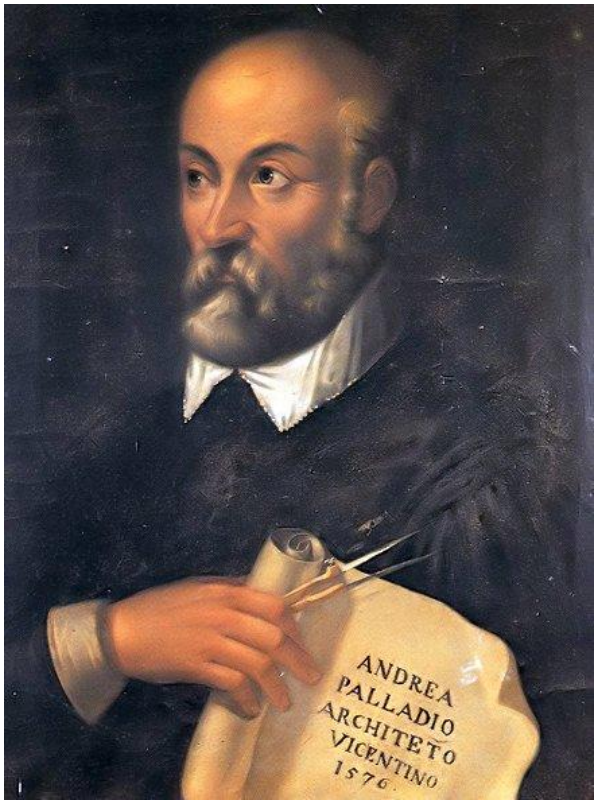
- Design component model of automated teller machine (ATM)
- Use all shown third-party components
- Add *required* and *provided* interfaces to **ATM component**



PALLADIO

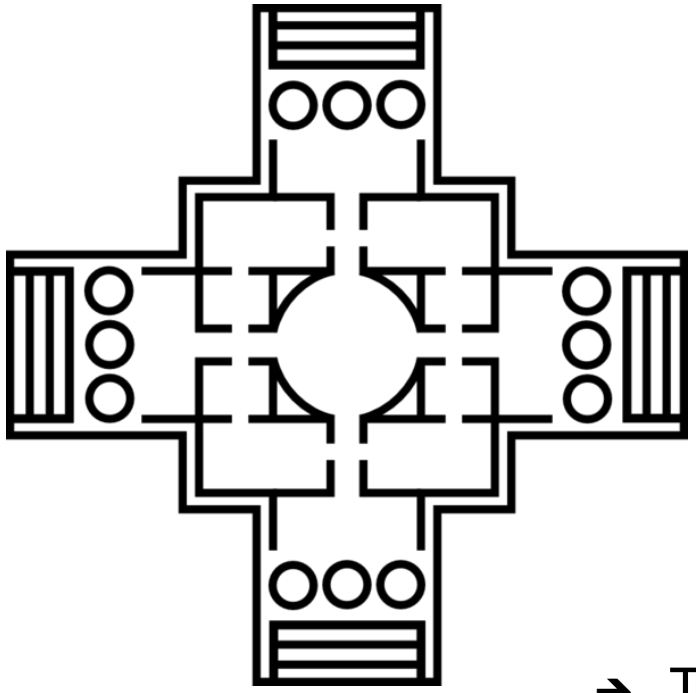
Andrea Palladio

- Andrea Palladio (30 November 1508 – 19 August 1580) was an Italian architect active in the Republic of Venice.
- Palladio, influenced by Roman and Greek architecture, primarily by Vitruvius, is widely considered the most influential individual in the history of Western architecture.
- All of his buildings are located in what was the Venetian Republic, but his teachings, summarized in the architectural treatise, The Four Books of Architecture, gained him wide recognition.



[Wikipedia]

Palladio Component Model (1)



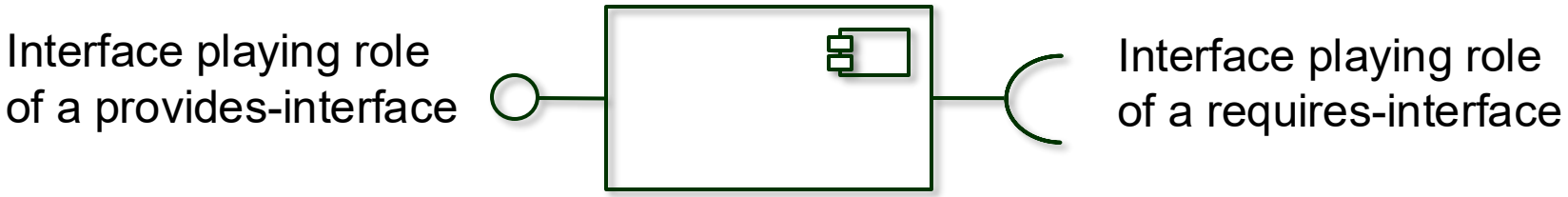
- Palladio Component Model (PCM) is a domain specific modelling language (DSL)
- Designed to enable early performance predictions for software architectures
- Aligned with component-based software development process

➔ Targets at –

- performance prediction for component-based software architectures
- business information systems

Contractual Use of Components

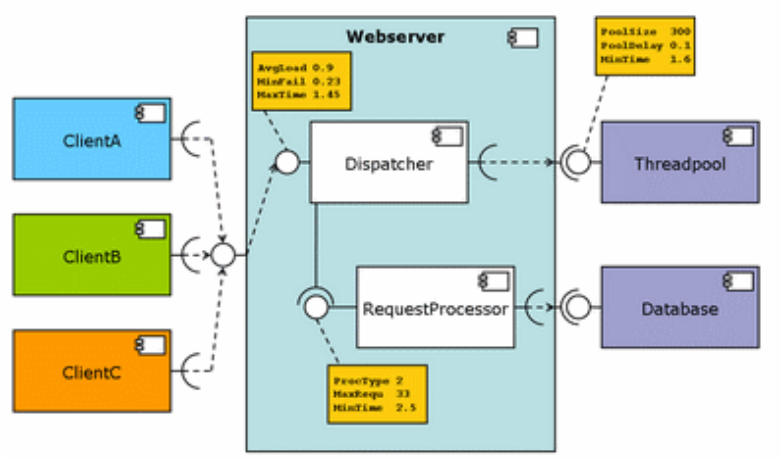
- Like design by contract,
  - but now for components
    - Classical DbC: methods (Bertrand Meyer)
    - Hoare-Triples: statements (Sir Tony Hoare))

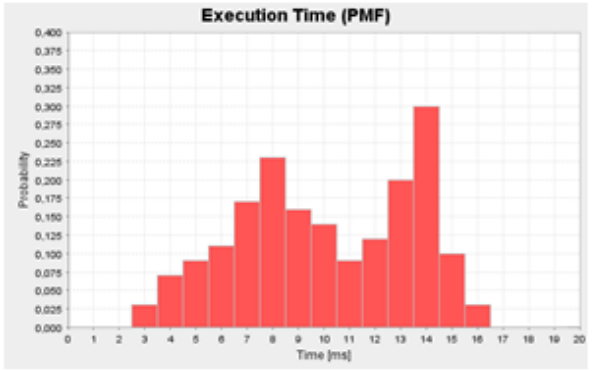


- Pre-Condition: what does the component expect from the assembly context to offer its services (as specified in the ***requires-interface(s)***)
- Post-Condition: what does the component offer, if precondition is fulfilled (as specified in the ***provides-interface(s)***)

General Idea

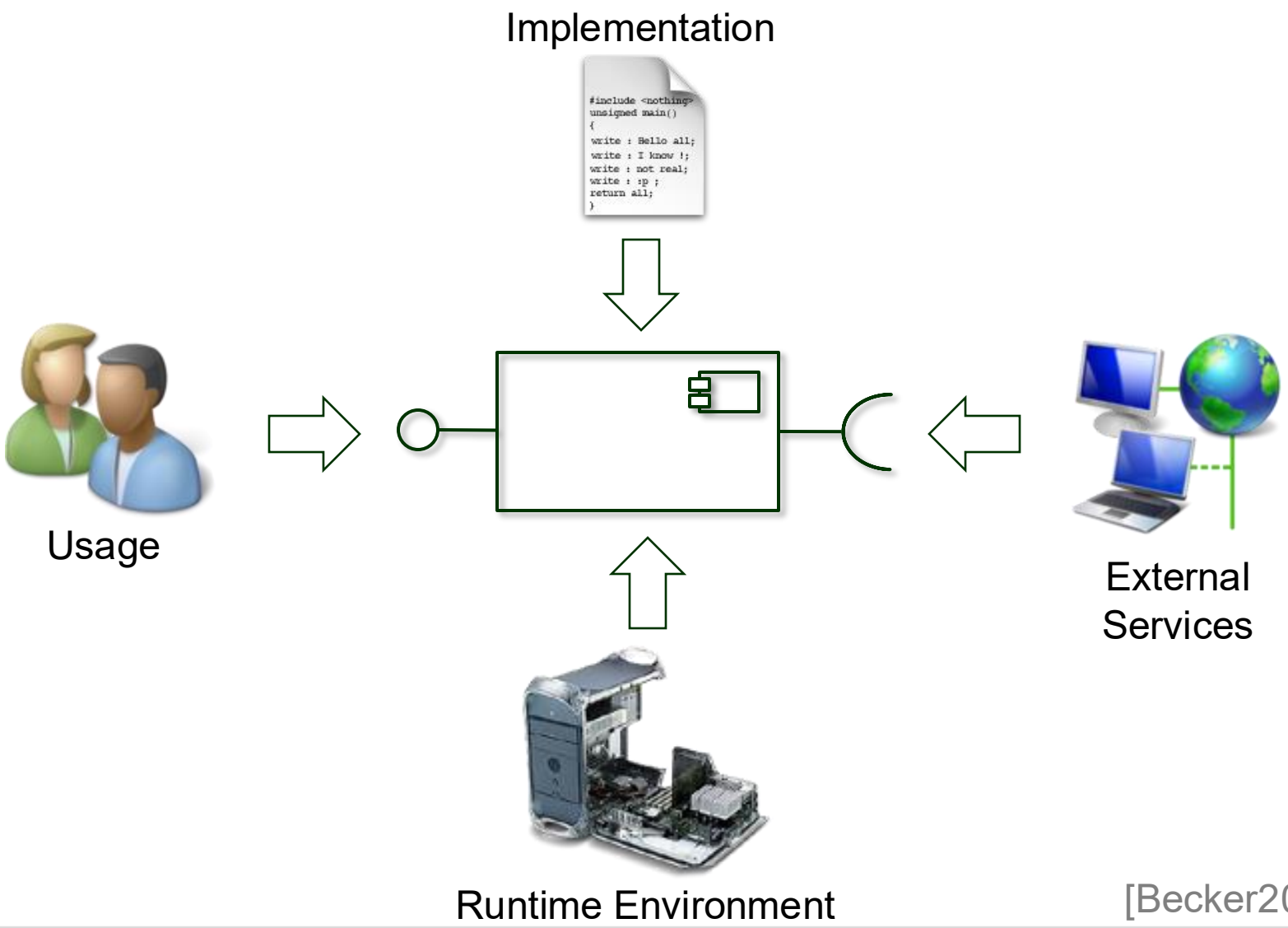
- Prediction of quality properties on a model base
  - for systematic design of software systems
  - performance, reliability, costs





- Derive performance metrics from the models using
  - analytical techniques and
  - simulation

Component Performance (1)



[Becker2006a]

Component Performance (2)

- *All* influence factors are made explicit in the PCM
- Required for conceptually clear components
- Supported context changes:
  - *Allocation context* → execution system (hardware / middleware / virtual machines)
  - *Usage context* → usage profile
  - *Assembly context* → “wiring” (other components fulfil a required services)

  → Explicit parameters in the PCM

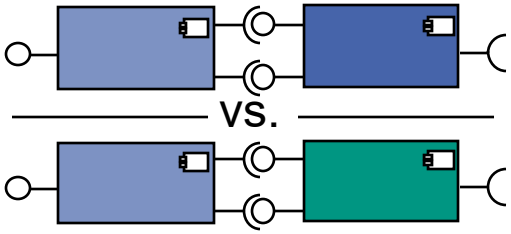
Supported Context Changes


vs.


Changing hardware
Sizing / scalability / relocation

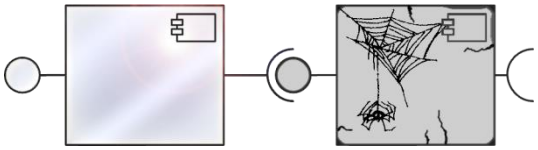

vs.


Changes in usage profile
Users interact differently with the system

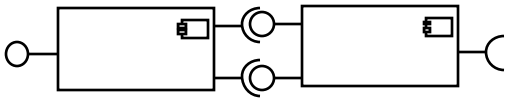


Changes in assembly context

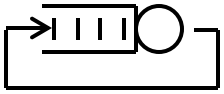
Why Creating Models?



Extensions of legacy software systems



Performance predictions at design time

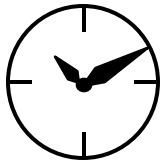


$$\sum_{i=0}^{N} p_l(i) \left(\circledast_{j=1}^{i} f_P \right)(t)$$

Analytically solvable

Performance jumps

Critical contention levels



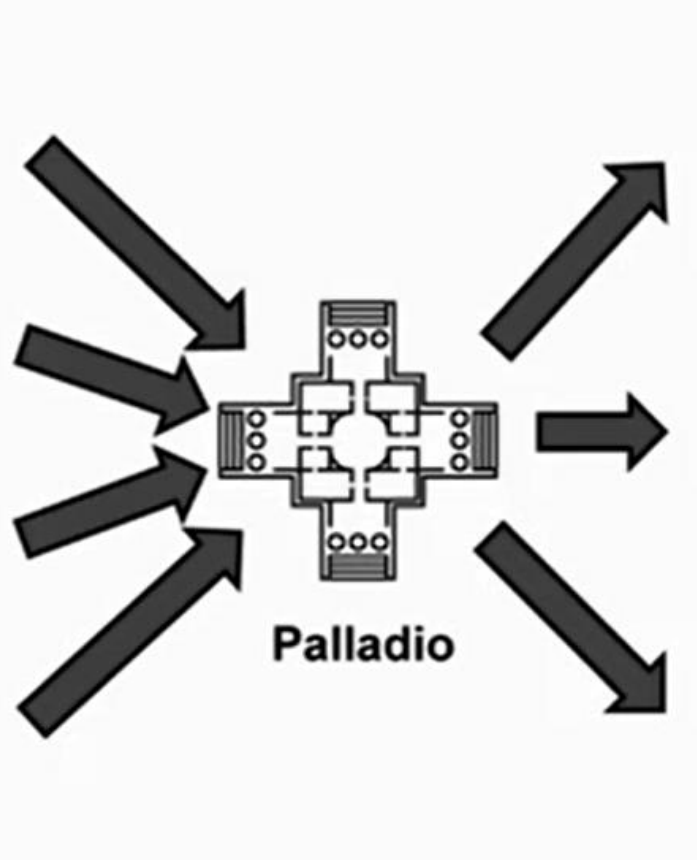
Simulation

Faster than “real” execution

Component Description

- What are the intuitive inputs and outputs for a performance prediction model?

?

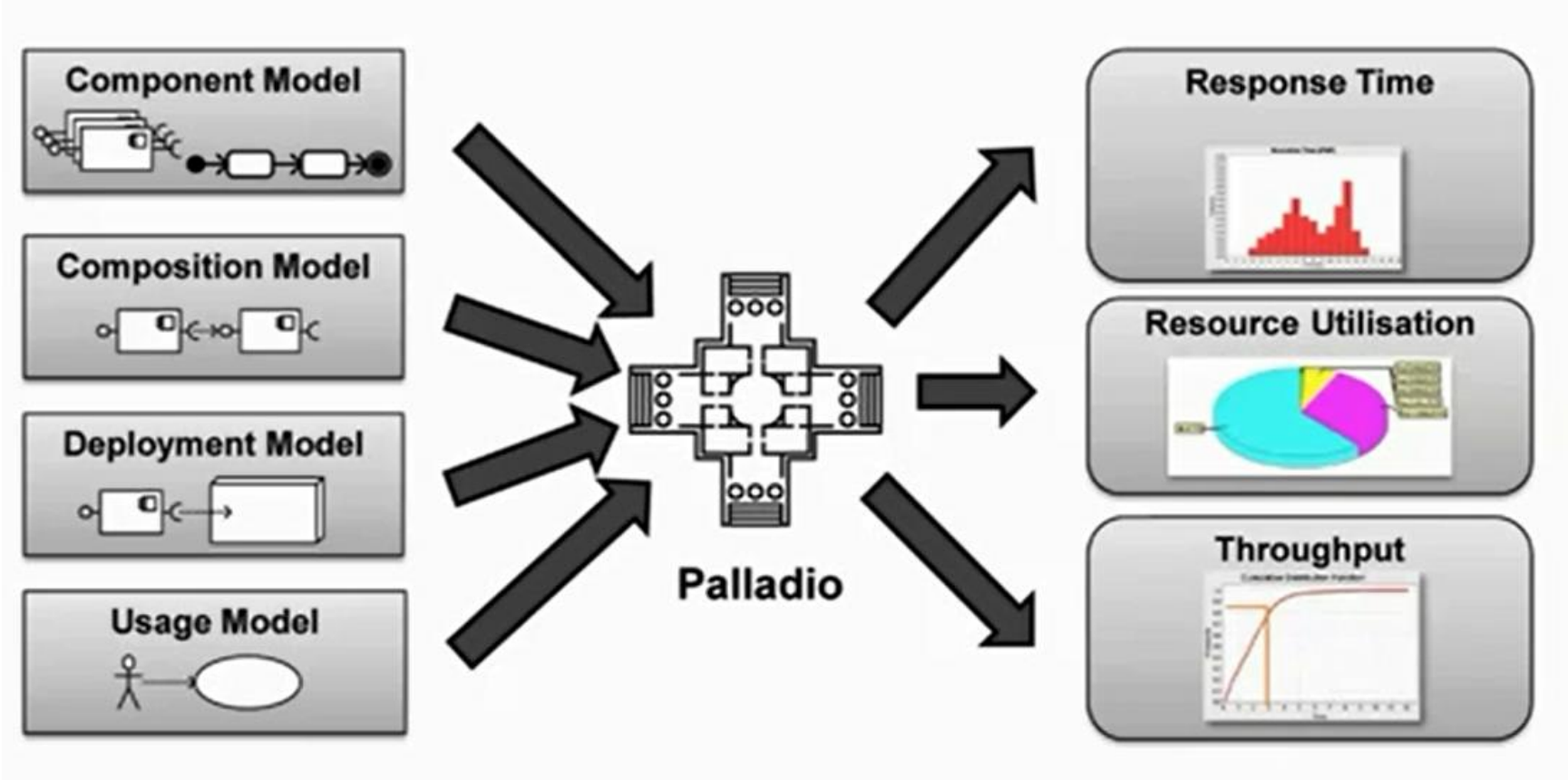


?

also see: http://www.youtube.com/watch?v=H0Gj-kdGhRs

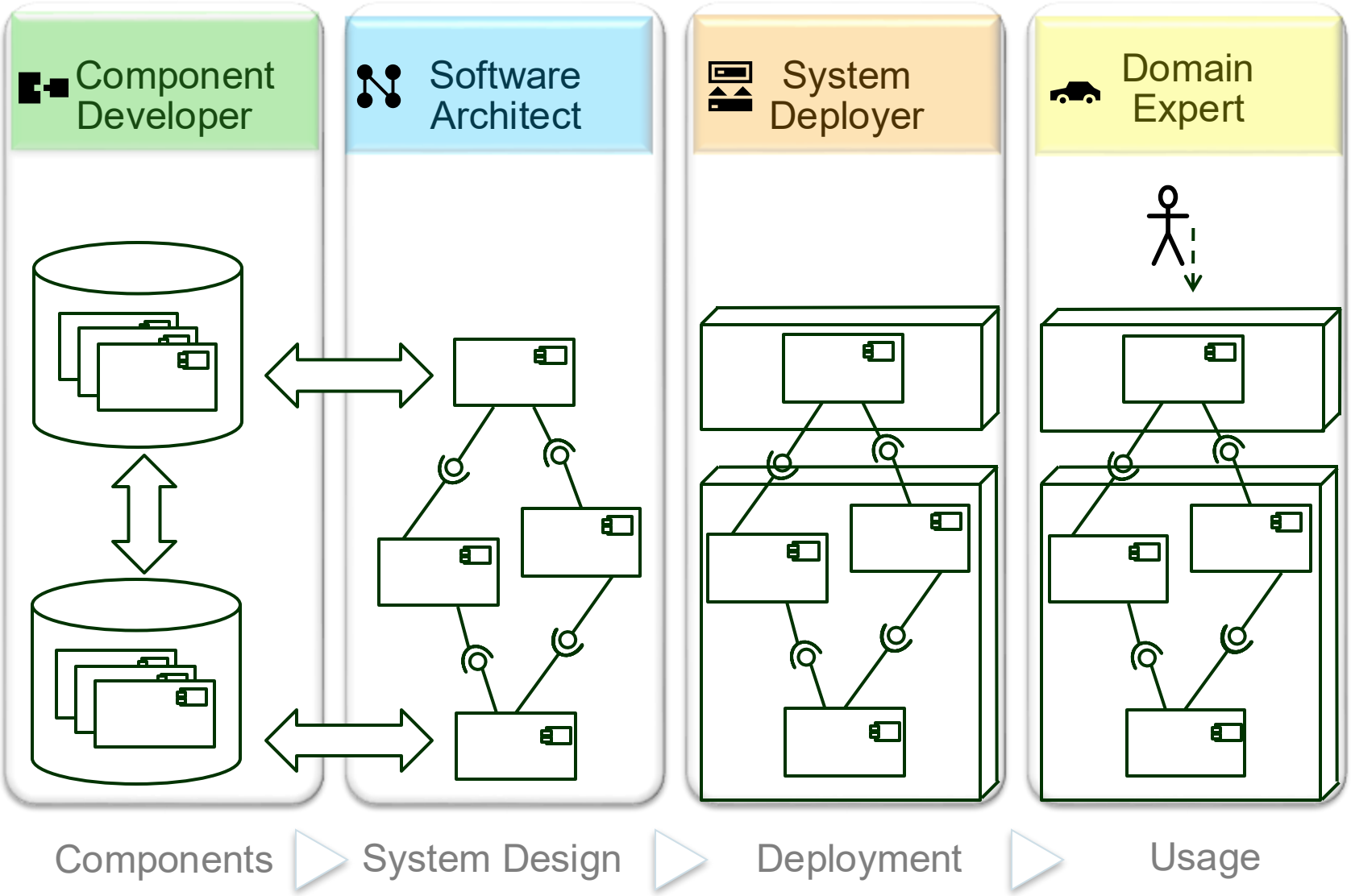
Component Description

- What are the intuitive inputs and outputs for a performance prediction model?

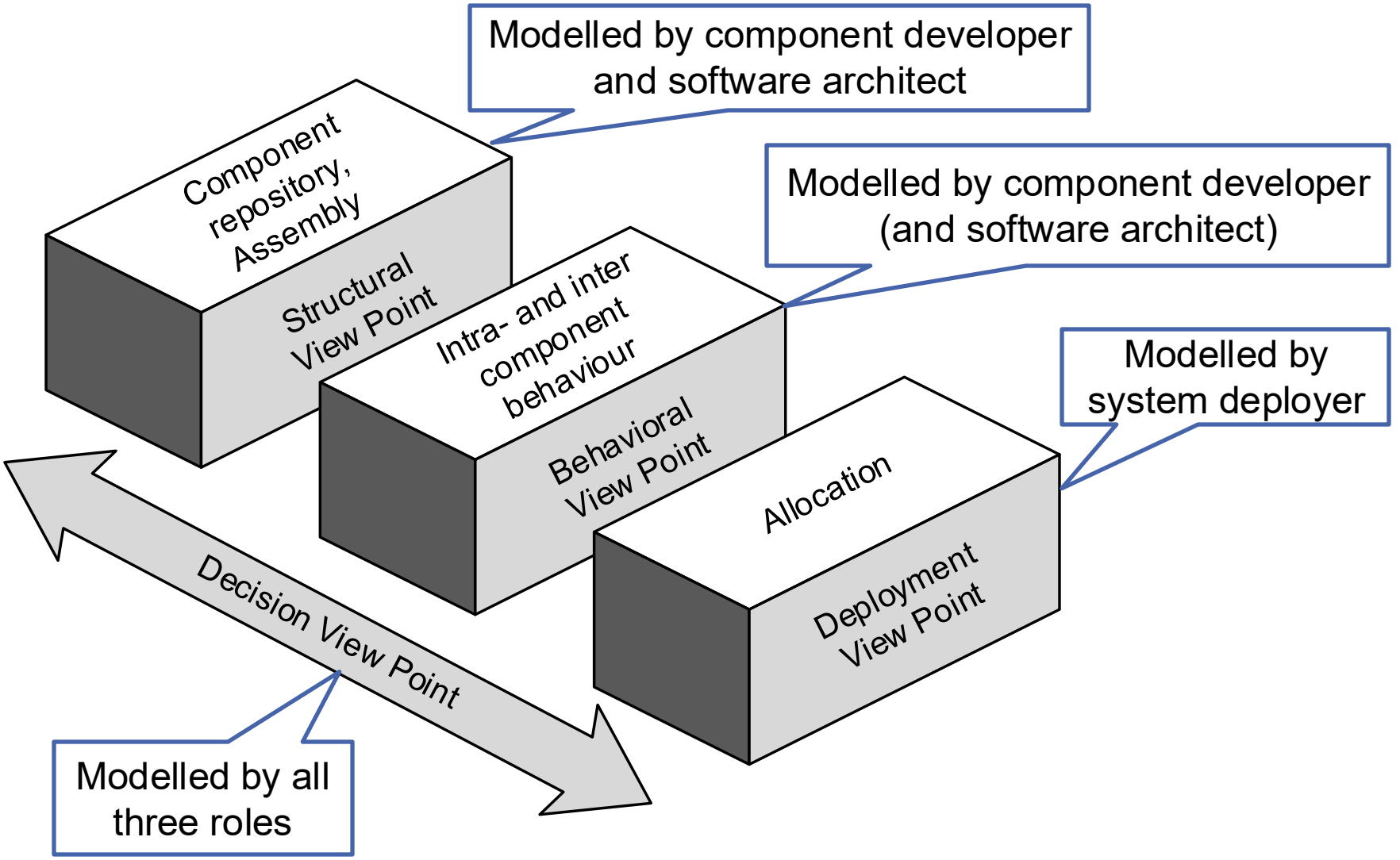


also see: http://www.youtube.com/watch?v=H0Gj-kdGhRs

Models and their Creators

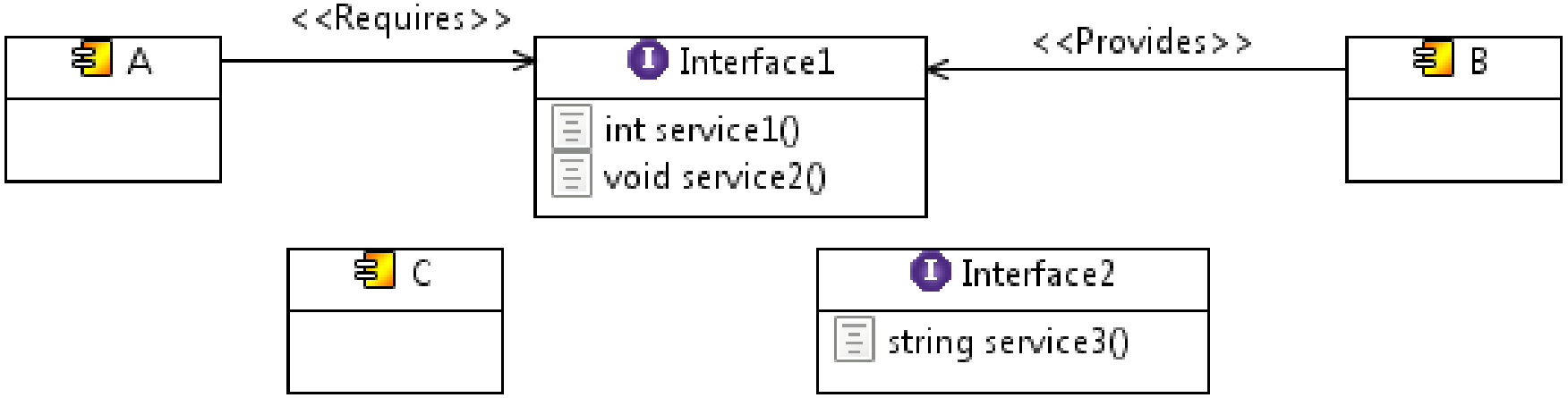


Views and View Points in Palladio



Component Description

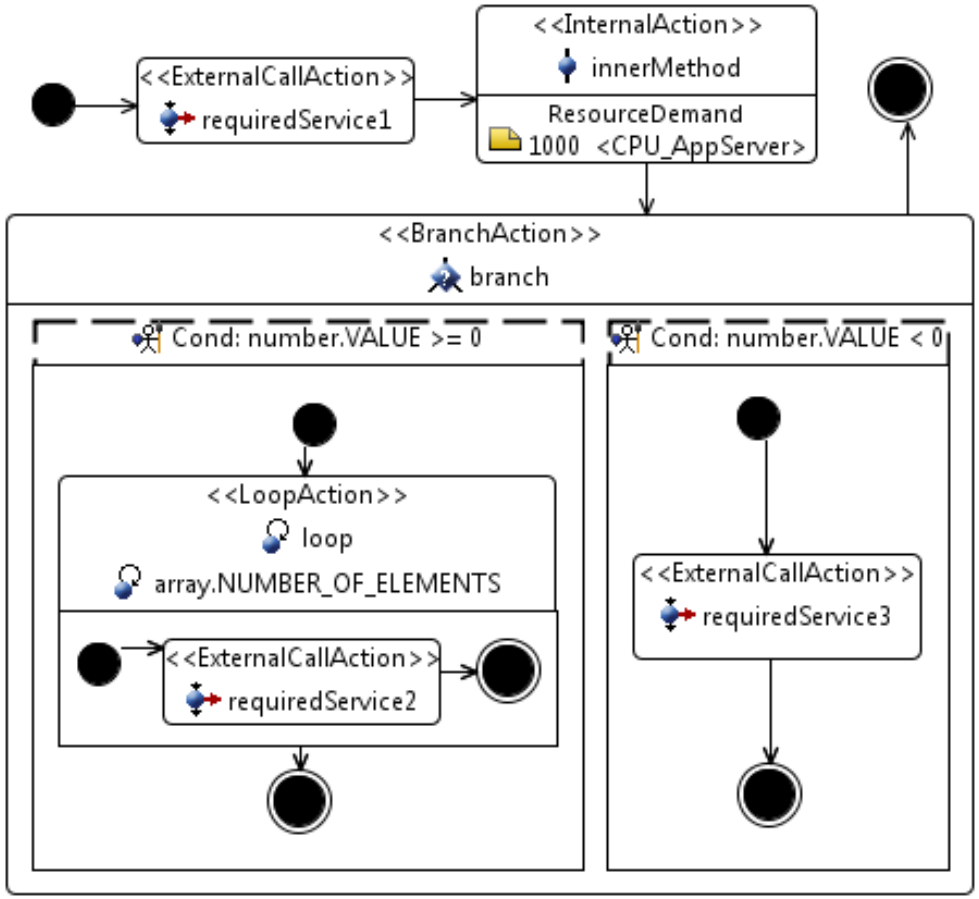
- Component interfaces need to be described
- Created components are stored in a repository



Component Developer

Behaviour Specification

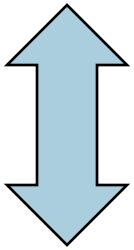
- Component behaviour needs to be described in so-called Service Effect Specification (SEFF)



Component Developer

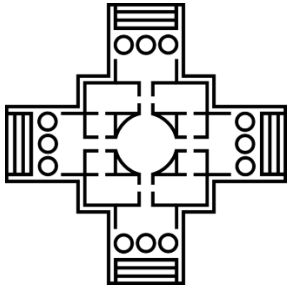
Component Performance?

- Component developer supposed to create readily composable component specs
  - reusable in different contexts unknown to component developer



Conflict!

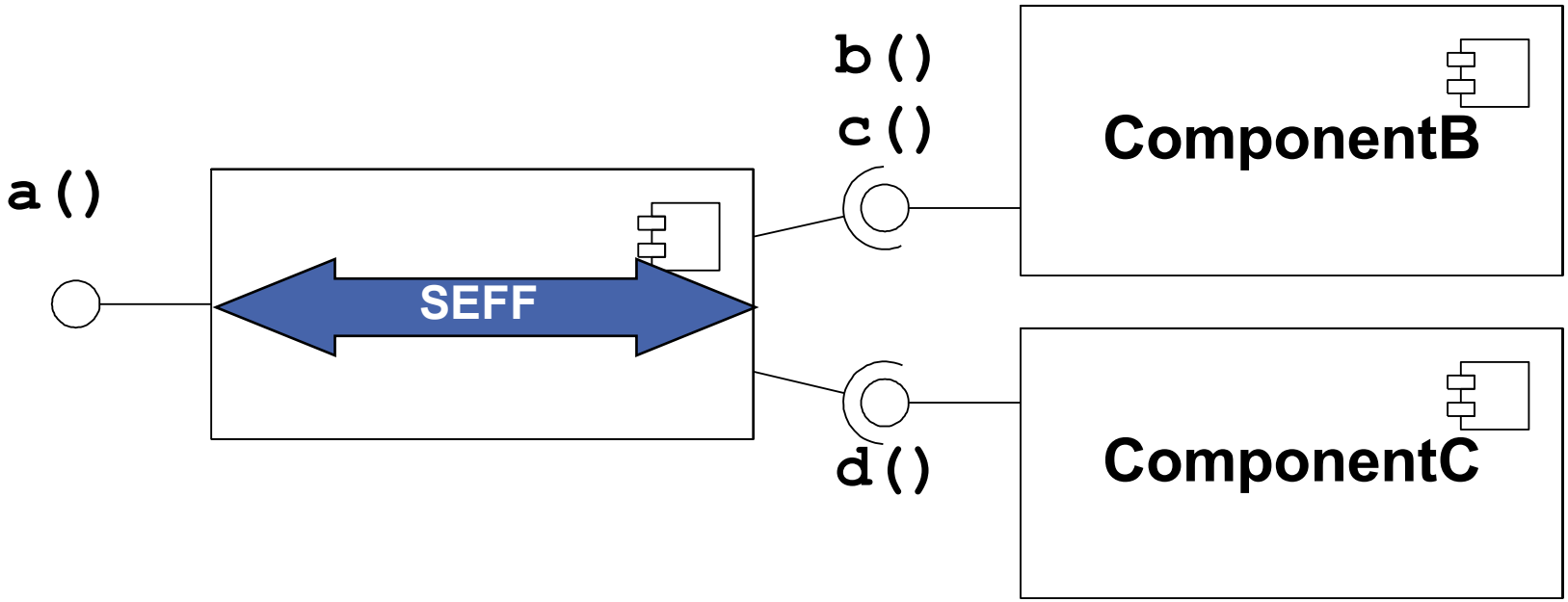
- Component performance depends on three influence factors
  - determined by context
- Solution: Roles, SEFF and parameterisation



Service Effect Specification

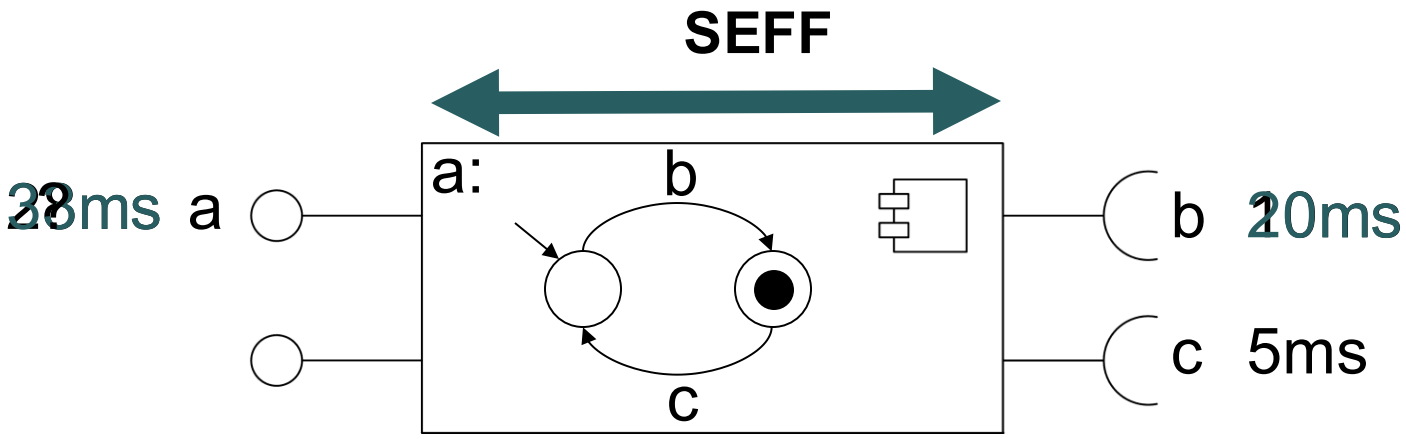
- Description of the external visible actions of a component's provided service
- Abstraction of internal behaviour
- Describes relationship between provided component side and required component side
- Can be parameterised by variables

Service Effect Specification: Idea



Component Developer

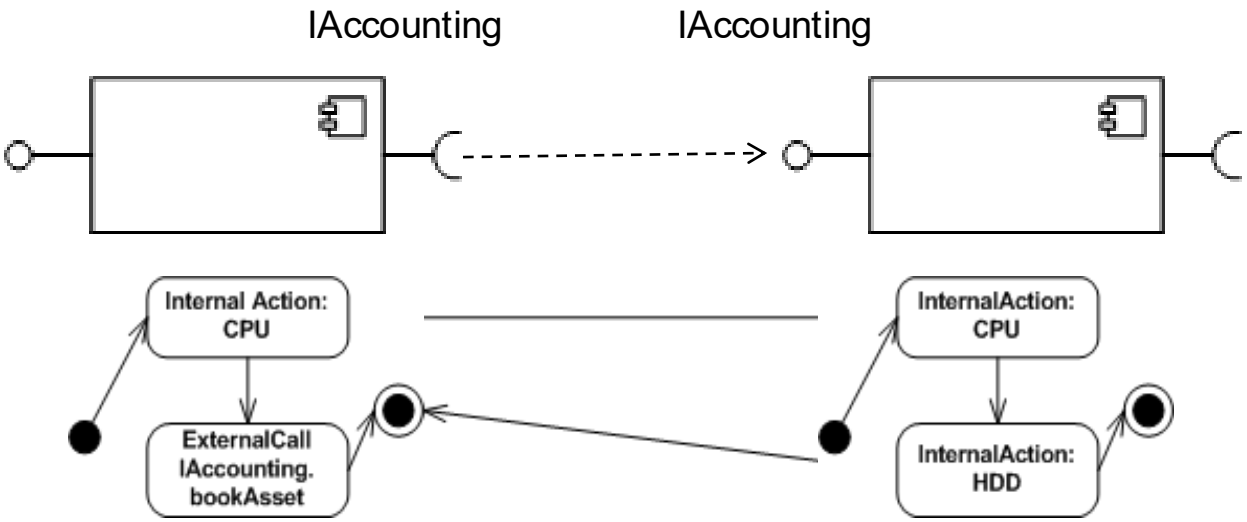
Service Effect Specification: Idea



Component Developer

Independent from External Services

- Explicit modelling of external service calls
- Binding to interfaces only
  - actually called service depends on assembly

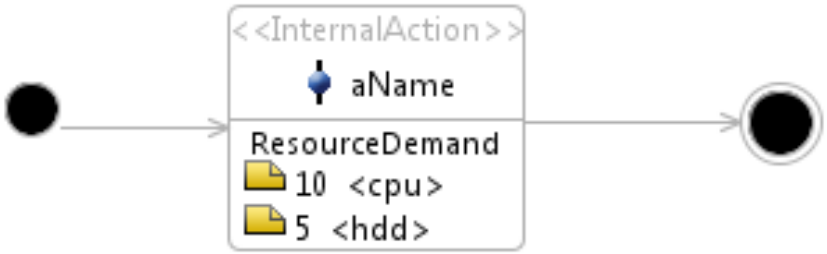


- After assembly, SEFFs could be combined to describe the whole system's behaviour.

Component Developer

Parameterisation of Resource Environment

- Internal actions specify demand of a component on processing resources
- Expressed in time values ?
  → No, shall be resource environment dependent
- Expressed in abstract units



- Timing values can be derived after resource environment is known
  - After resource environment is specified by deployer

Component Developer

Parameterisation of Usage

- Resource demand and behaviour depend on usage
- Examples:
  - compression of a file depends on file size
  - number of loop iterations depend on collection size
- Abstraction of user data
  - performance relevant information
  - `Array.NUMBER_OF_ELEMENTS`, `file.BYTESIZE`, `request.TYPE`, ...
- Usage must also be propagated

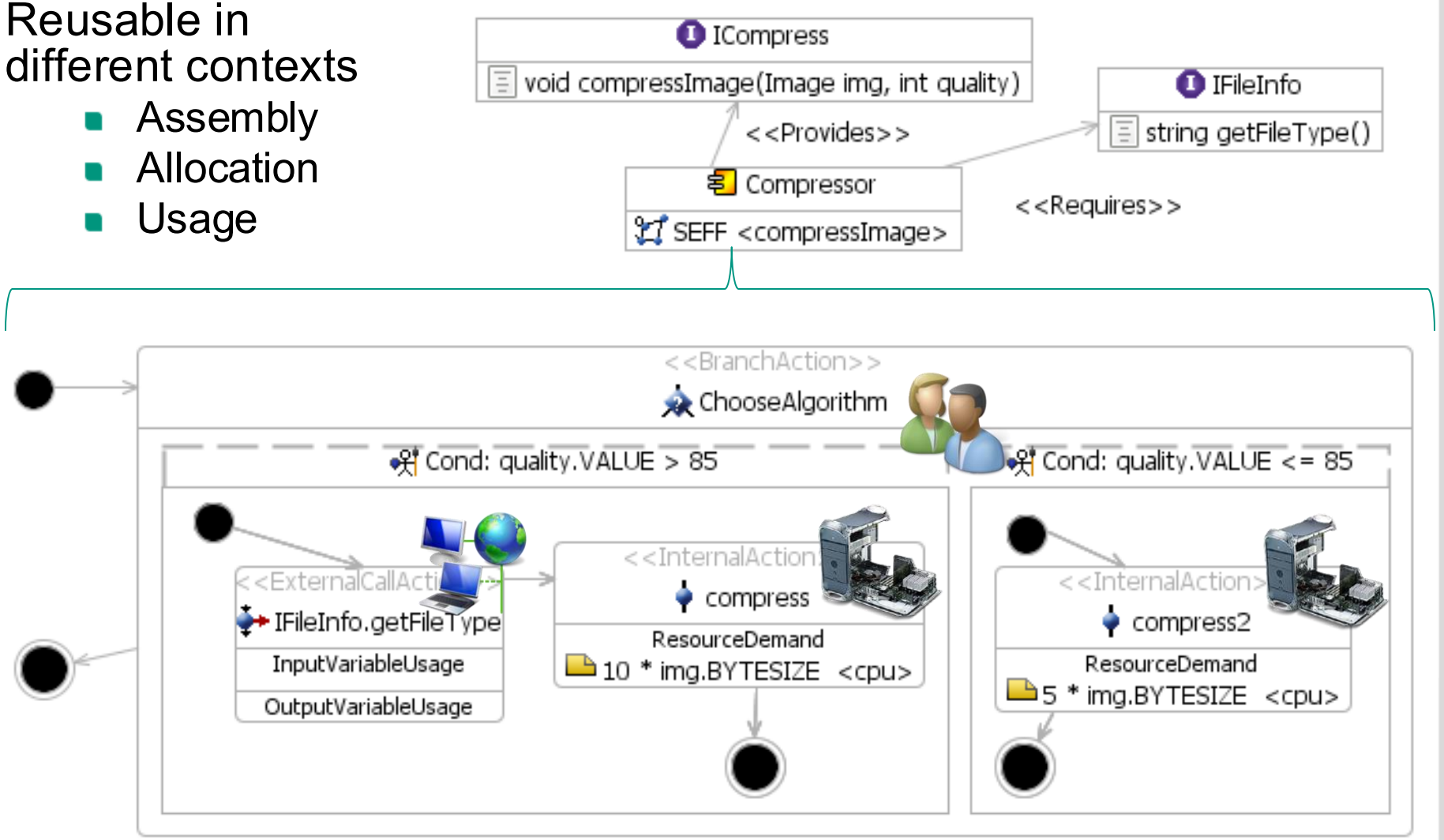
Parametrisation Combined

Component Developer

Karlsruhe Institute of Technology

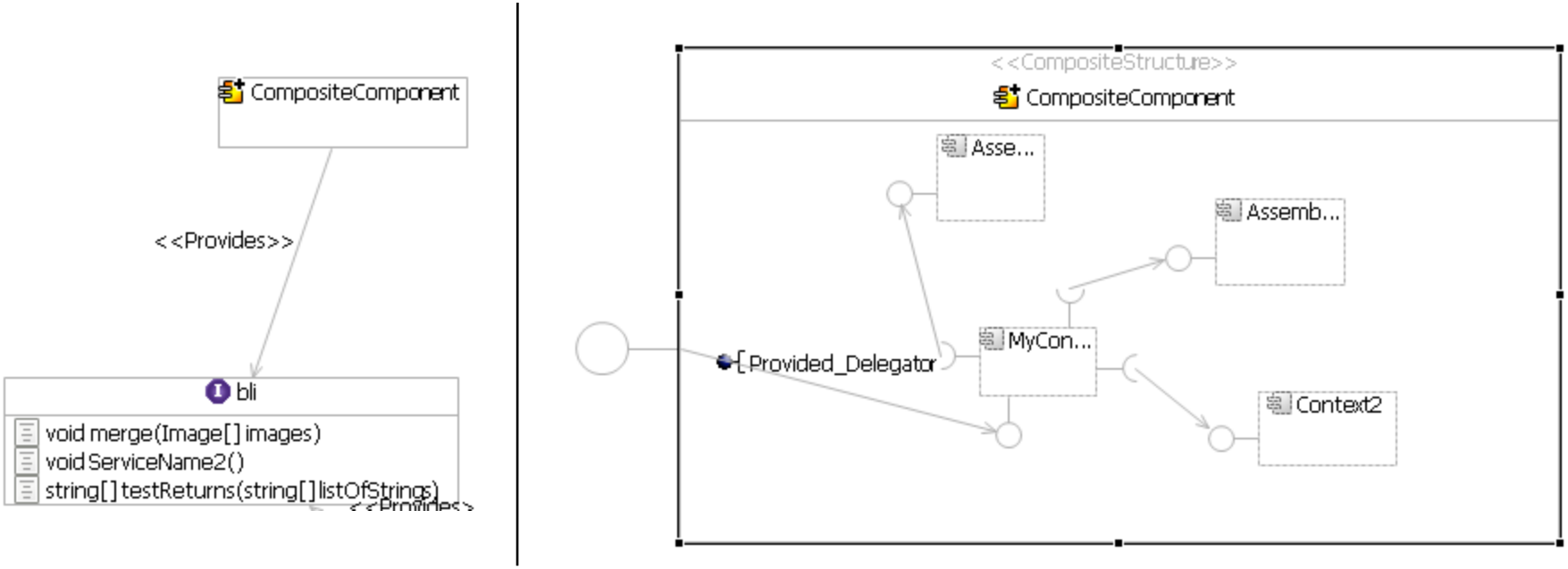
Reusable in different contexts

- Assembly
- Allocation
- Usage



Composite Component

- Composed from *basic* Components and/or other *composite* Components



Component Developer

Component Developer: Tasks

PCM Tasks

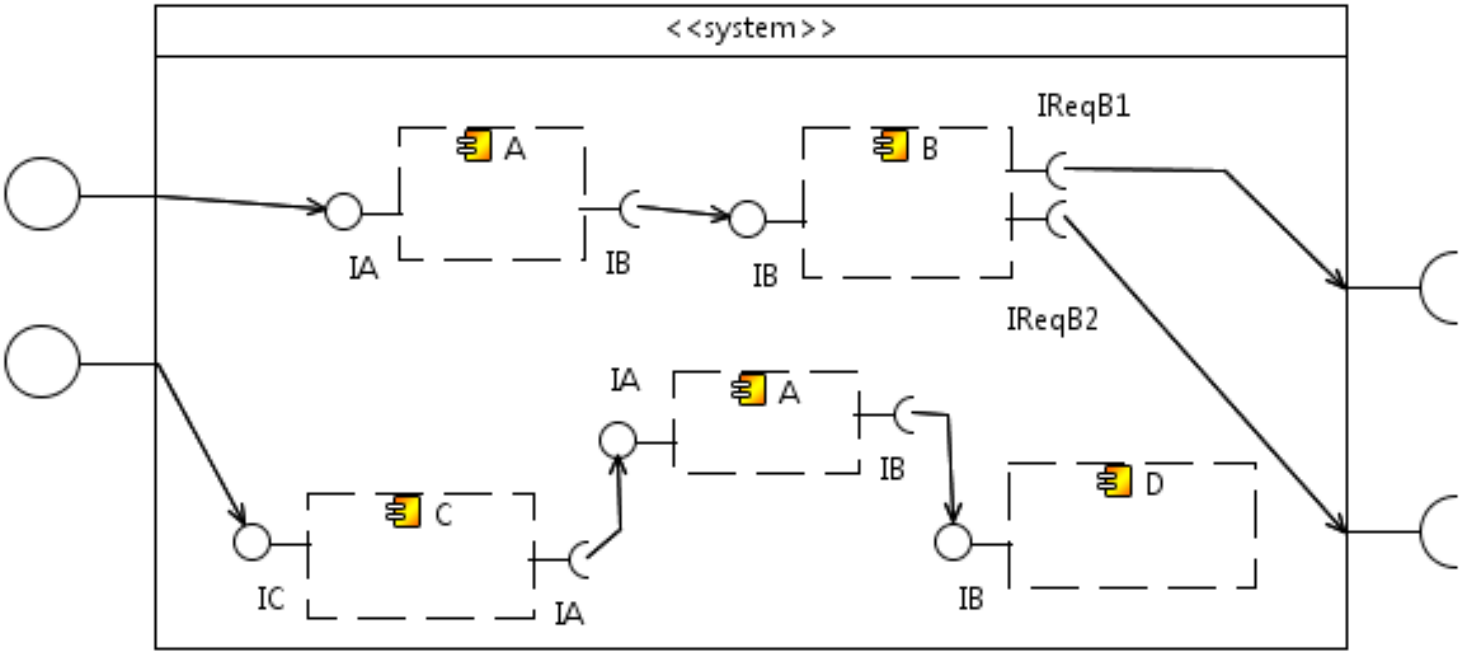
- specifies components & interfaces
- specifies data types
- builds composite components
- creates parameterised service effect specifications
- stores modelling & implementation artefacts in repositories

General Tasks

- implements components
- tests components
- maintains components

System Composition

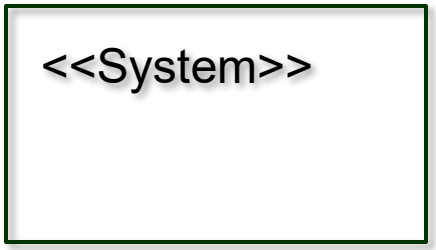
- System is composed of components from repository



Software Architect

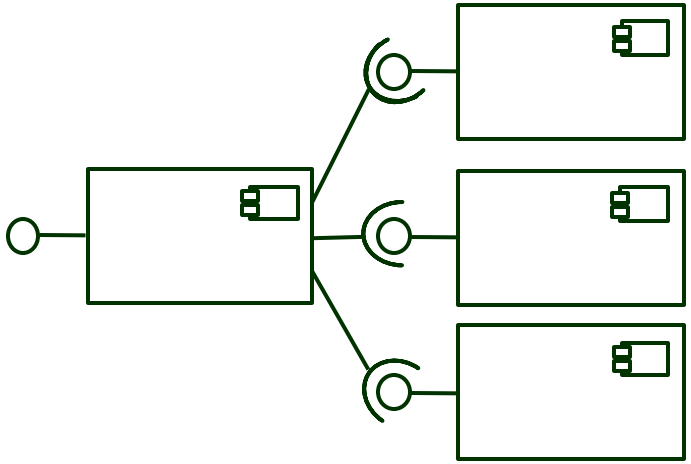
System Model

- Models the component-based architecture to be analysed
- May include components from different repositories
- Provides an interface for users
- Excludes uninteresting services and connects to them via system required interfaces
- Is a prerequisite for the system deployer to allocate the components



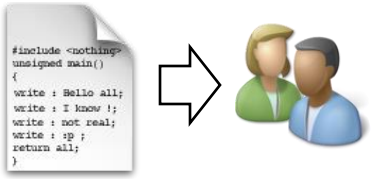
Software Architect: Tasks (1/2)

- Specifies an architecture (system model) from existing components and interfaces
- Specifies new components and interfaces
- Uses architectural styles and architectural patterns
- Analyses architectural specification and makes design decisions



Software Architect: Tasks (2/2)

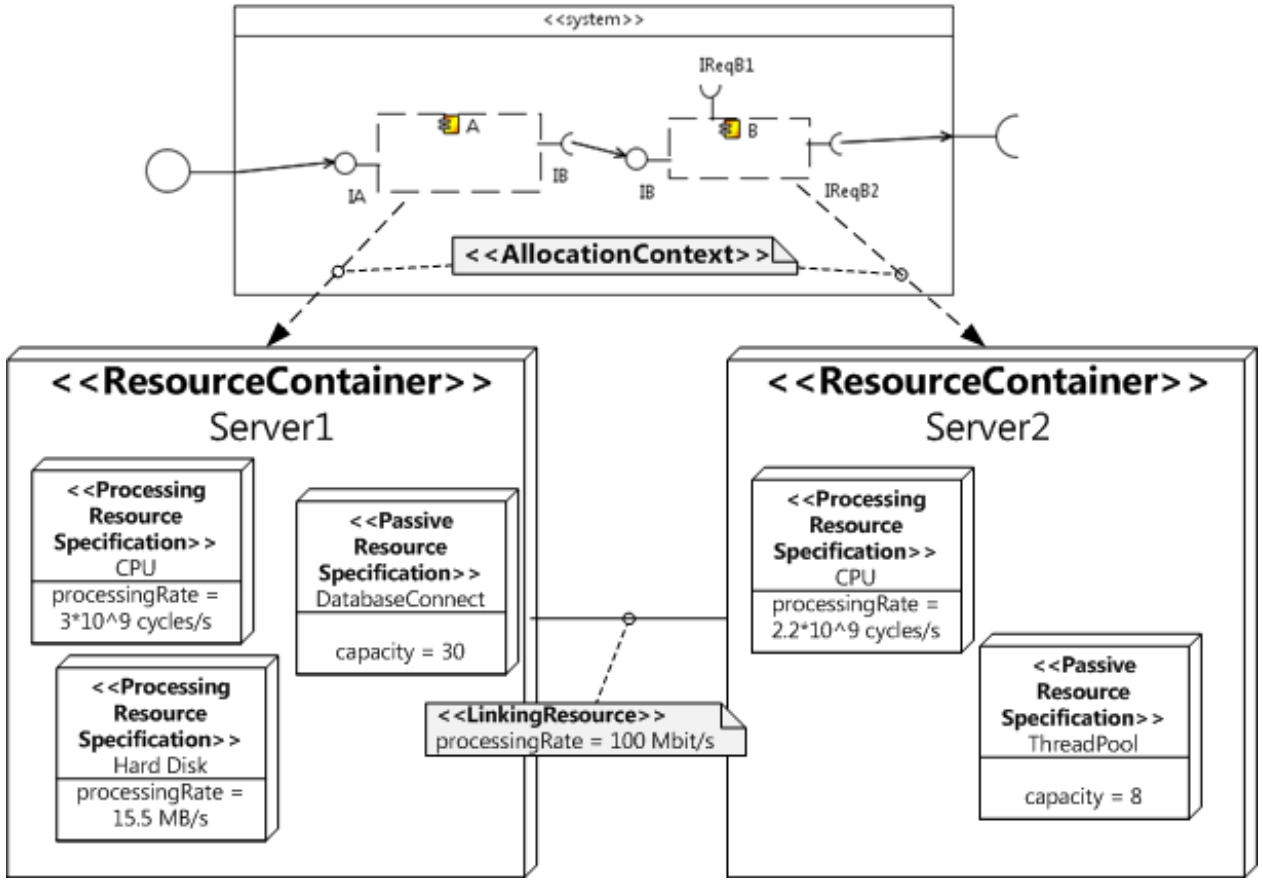
- Conducts performance prediction based on architectural specification
- Delegates implementation tasks to component developers
- Guides the whole development process



Software Architect

Resource Description

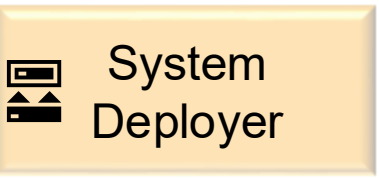
- The deployment environment of the system needs to be specified



System Deployer

Resource Types

- Abstract specification of resources
  (e.g. CPU, HD, Net)
- Why?
  - concrete resources (e.g. 2 GHz CPU, 20 MB/s HD, 1 Gbit/s Net) unknown during component specification and implementation
- Thus: component developers provide
  SEFF specifications referring to resource types
- Once the concrete resource environment
  is specified, timing values can be derived



Resource Types in PCM

CPU

HD

Network

Memory

System Deployer

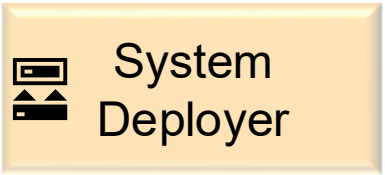
System Deployer: Tasks

PCM Tasks

- models the resource environment
  (e.g., middleware, OS, hardware)
- models the allocation of components
  to resources

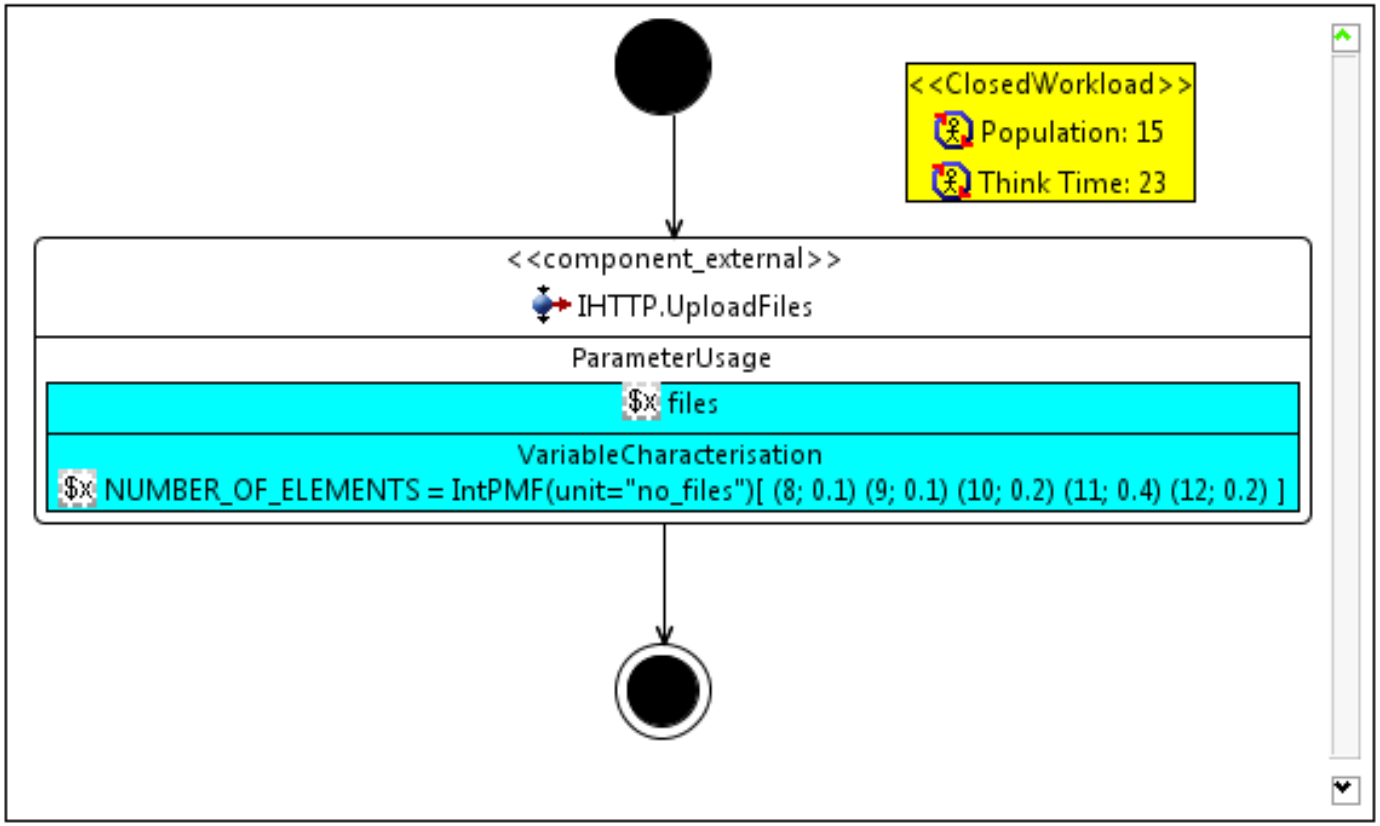
General Tasks

- sets up the resource environment
  (e.g., installing application servers, configuring hardware)
- deploys components on resources
  (e.g., writing deployment descriptors)
- maintains the running system



Usage Model

- Finally, we need to specify how the system is used



Domain Expert

Usage Model

- Models *user* behaviour, not component!
- Similar to SEFFs, but
  - does not refer to resources
  - does not refer to inner components of a system
  - does not model parametric dependencies
  - includes a workload specification
- Usage Model
  - *1…n* usage scenarios (1 per use case)
  - *1* workload per usage scenario

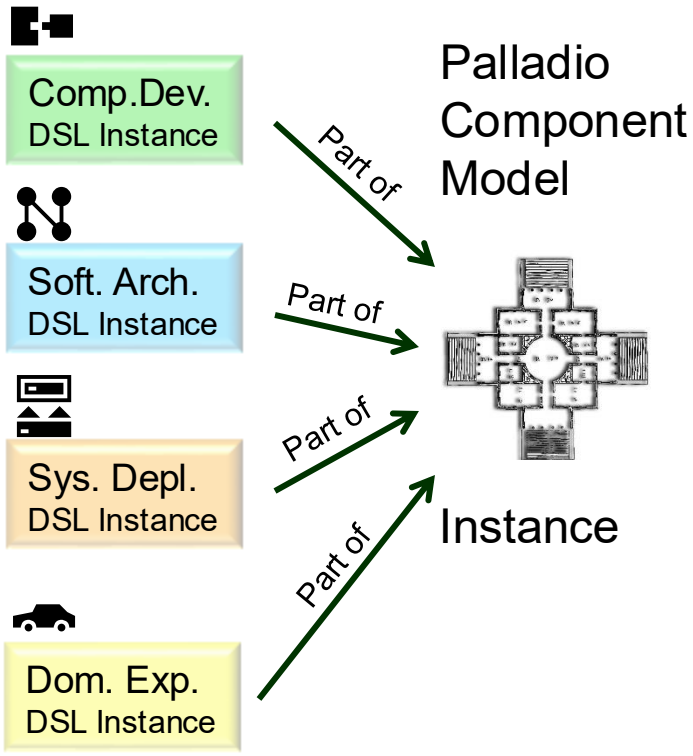
Domain Expert

- Familiar with the business domain

- PCM Task: Specifies user behaviour
  - number of users
  - user requests to the system
  - input parameters characterisations

Resolve Dependencies

- After the complete model has been built
  - Component developer: Components and SEFFs
  - Software Architect: Assembly
  - System Deployer: Resource environment
  - Domain Expert: Usage
- Black Box principle is maintained



Composing the Models

Assembly Context	Allocation Context	Usage Context
Specified by Architect & Developer:	Specified by System Deployer:	Specified by Domain Expert:

• **Horizontal Composition: Binding to other Components**
• **Vertical Composition: Encapsulation in Composite Components**
	• **Allocation to Hardware Resources**
	• **Configuration**
	  ○ Component, Container
	  ○ Communication
	  ○ Security, Concurrency
	  ○ …
		• **Usage at System Boundaries**
		  ○ User Arrival Rate
		  ○ Number of Users
		  ○ Request Probabilities
		  ○ Parameter Values

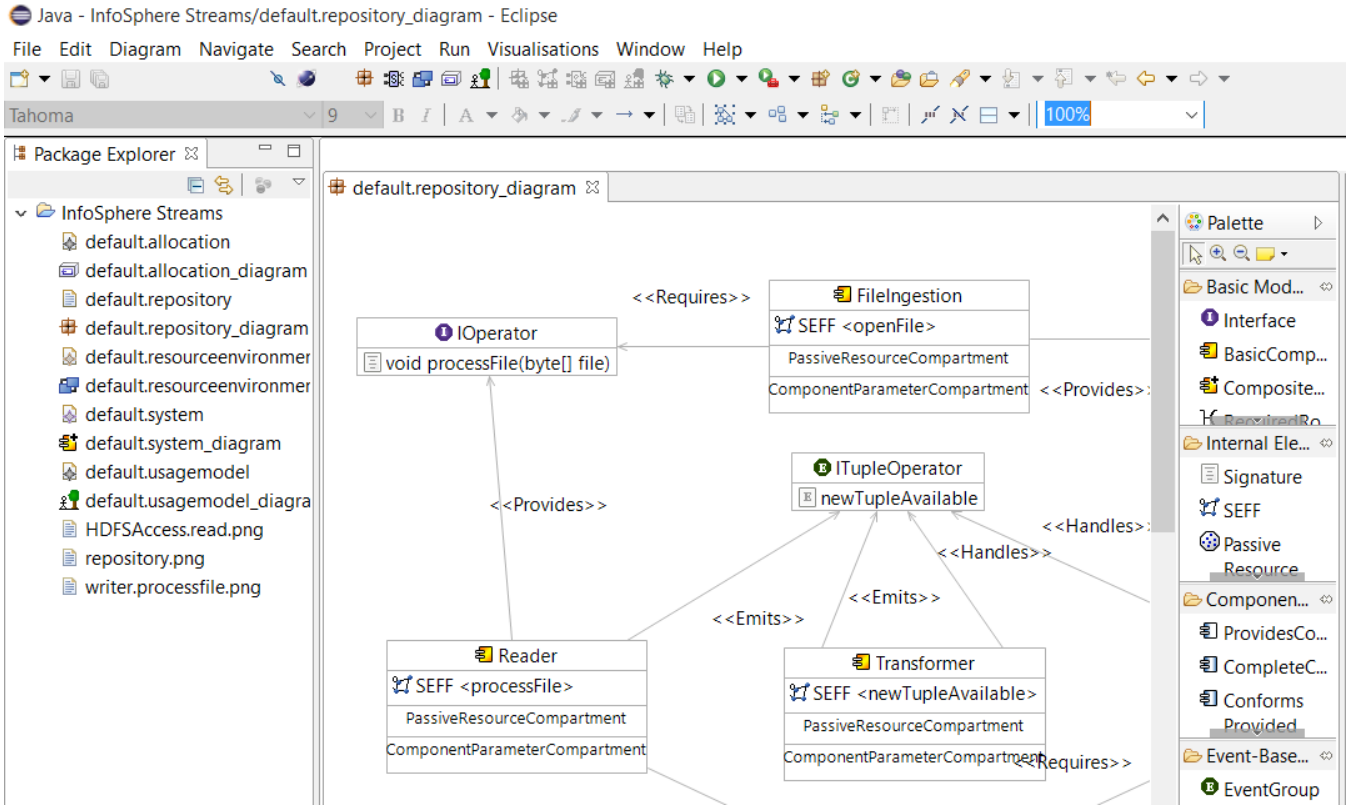
Computed by Tools:	Computed by Tools:	Computed by Tools:

• **Behaviour of the whole system**
  ○ “Overall SEFF”
	• **Allocation-dependent QoS Characteristics**
	  ○ Timing Values for Resource Demands
	  ○ Failure Probabilities
	  ○ …
		• **Usage inside Components**
		  ○ Branch Probabilities
		  ○ Loop Iteration Numbers
		  ○ Input/Output Parameters
		  ○ Usage-dependent Resource Demands

[Becker2008]

Tool Time: PCM Bench

- Supports whole component-based design process
- Analysis approaches provide hints on performance bottlenecks / issues



Component Developer

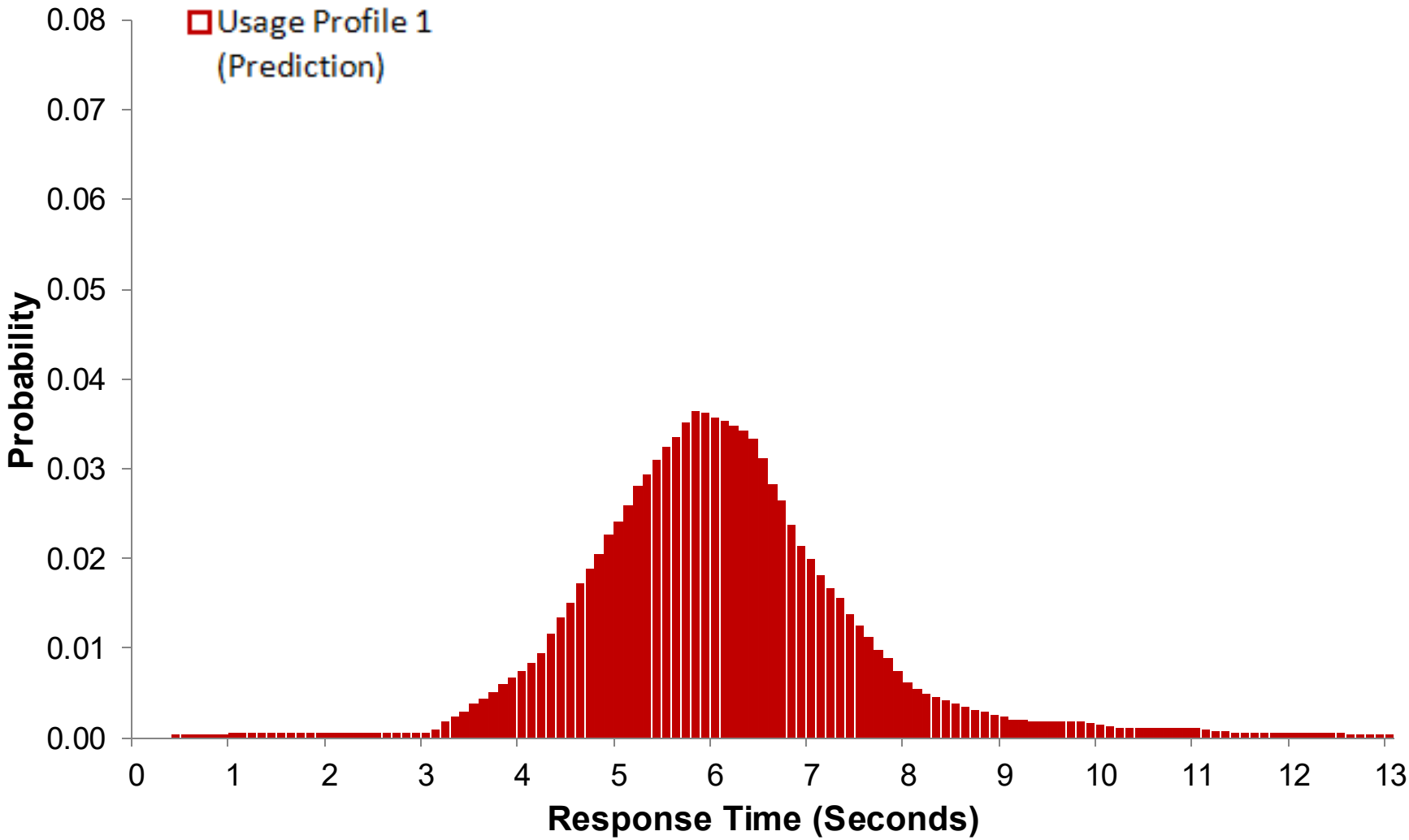
Software Architect

System Deployer

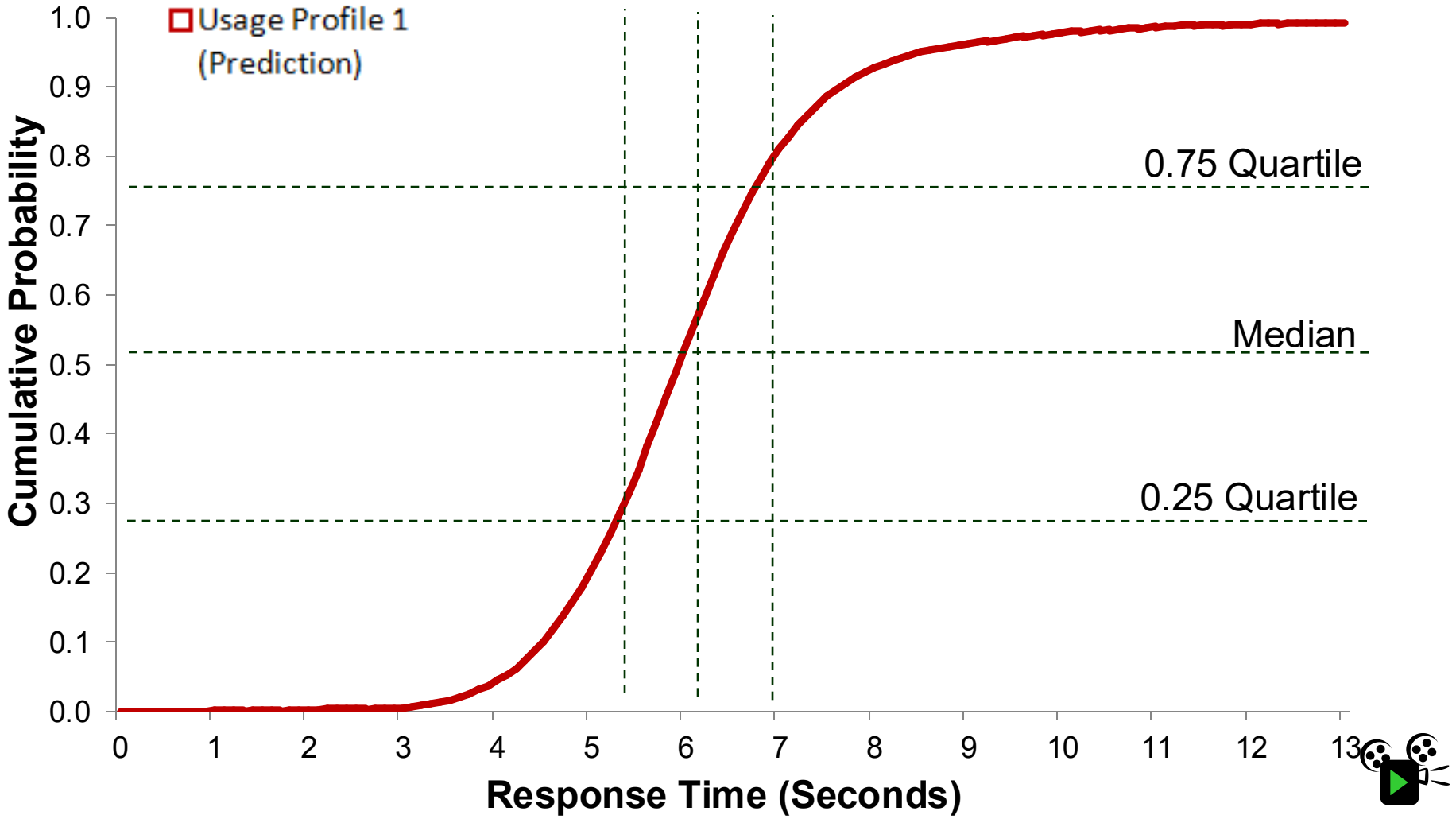
Domain Expert

see http://www.palladio-simulator.com

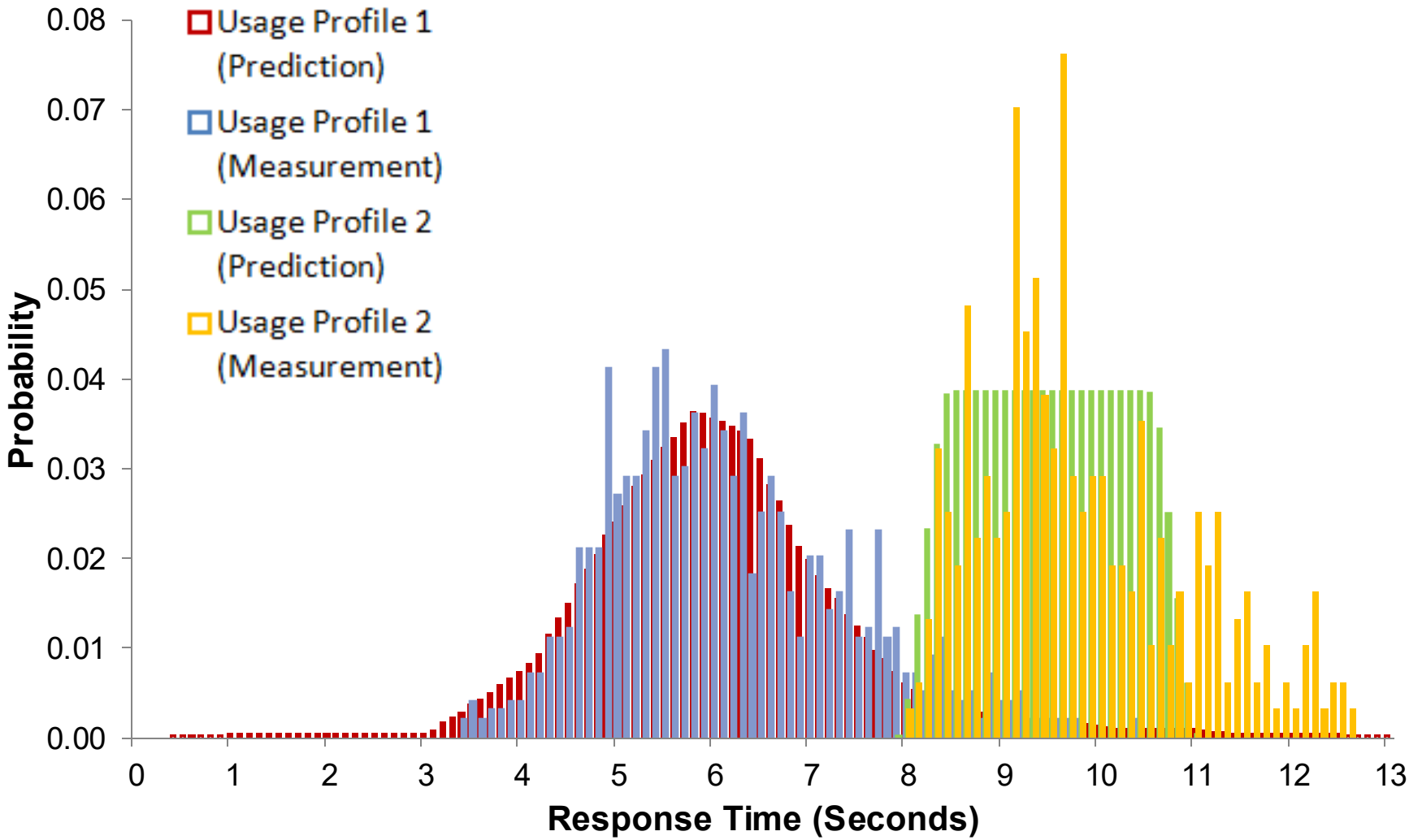
Result Example: Histogram



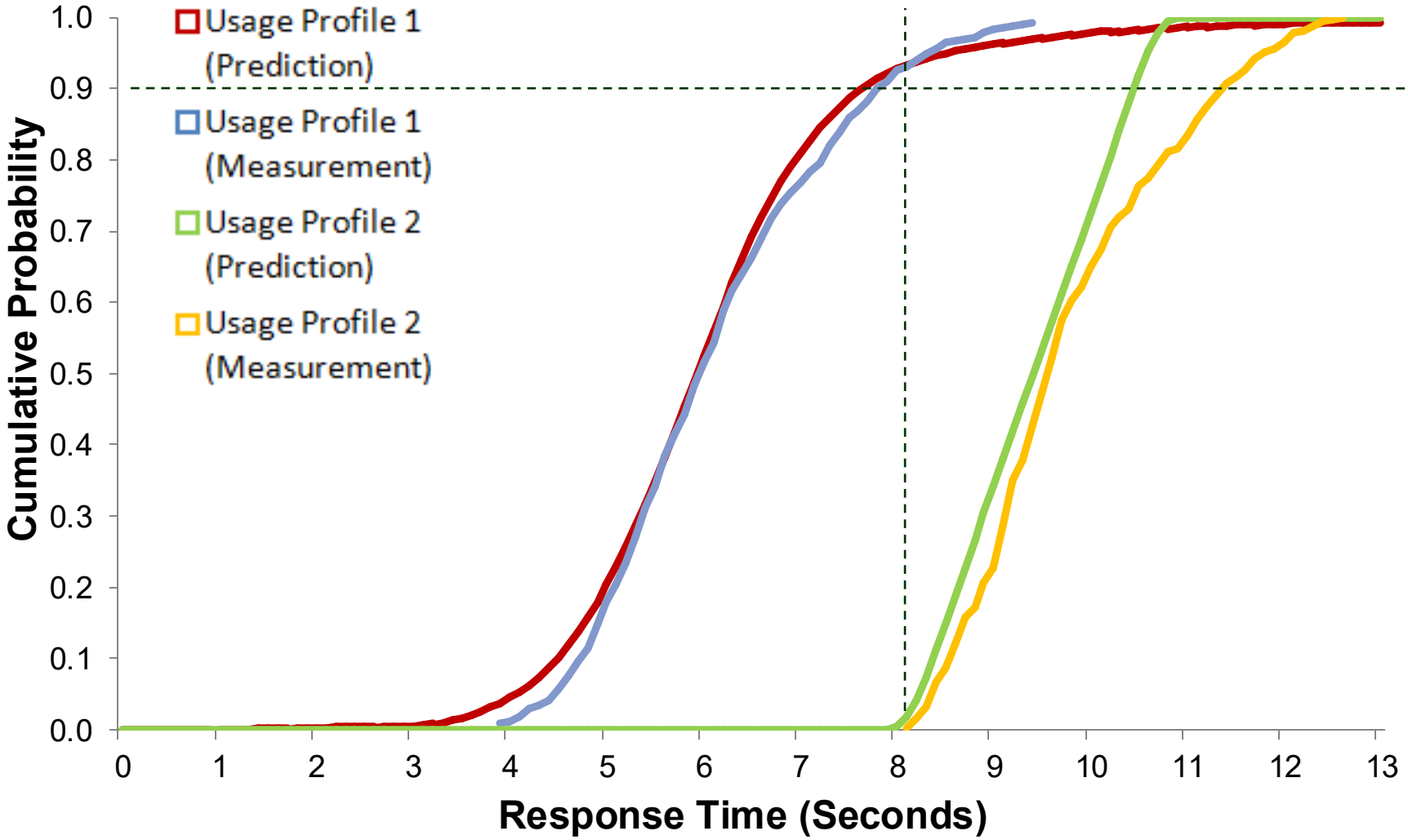
Result Interpretation



Comparison of Results I



Comparison of Results II



Lessons Learned: PCM

- Explicit performance influence factors in the model
  - code, external services, resource environment, usage
  - → compositional performance model
- Follows component definition of Szyperski
  - SEFF reusable in different contexts
- Clear component concepts
- Designed to support independent developer roles
- Keeps black-box principle of components at model level for assembly, allocation, and usage

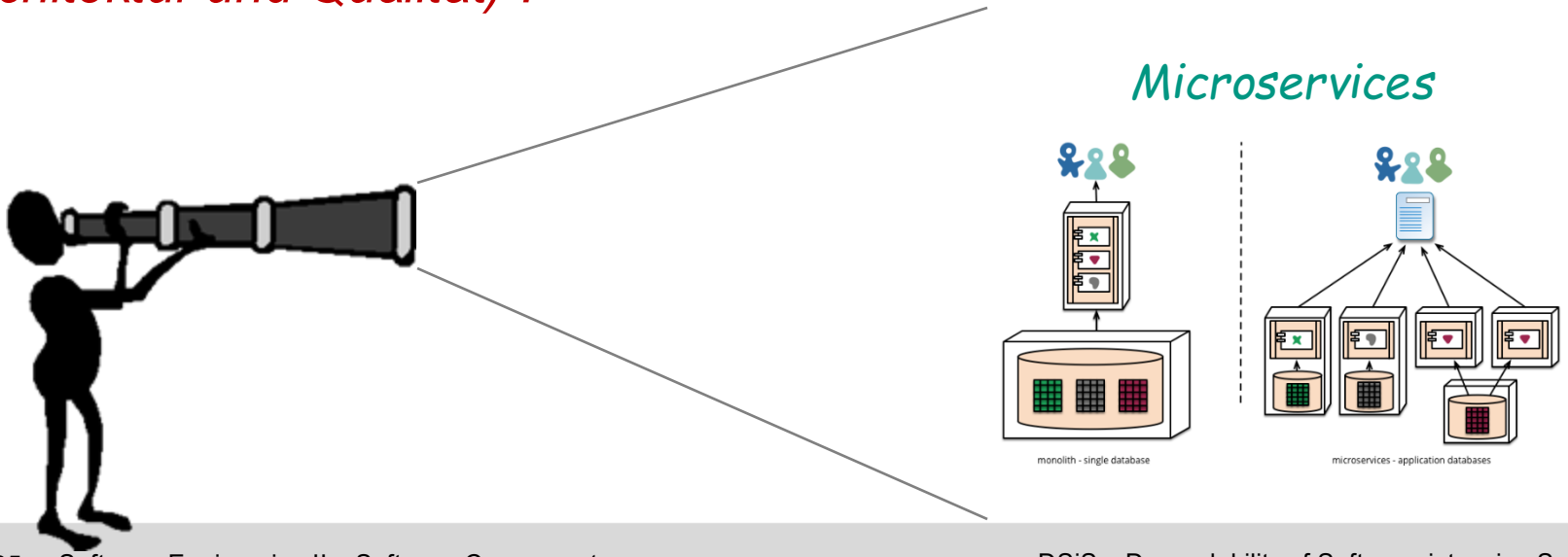
Overall Lessons Learned

- CBSE principles require
  - third-party use
  - readily composable
  - → separate developer roles
- Component performance depends on
  - external services
  - resource environment
  - usage
  - → SEFF and parameterisation
- Engineering approach to software development

Conclusion

- Components are supposed facilitate the composition of software systems
  - various component models are available
  - however, as so often, the concept is still overloaded

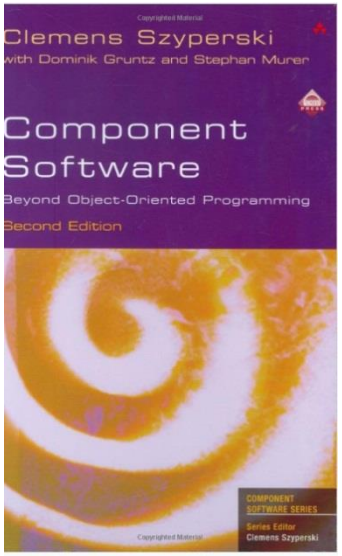
    and there is no „one size fits all“
- Composition at the evaluation level: Palladio Component Model
- *More to come in „Software Architecture and Quality (Software-Architektur und Qualität)”!*



References

Main references

- Ralf H. Reussner et al. *Modeling and Simulating Software Architectures — The Palladio Approach* MIT Press, to appear in 2016
- Becker, Koziolek & Reussner: *Palladio Component Model:* http://www.sciencedirect.com/science/article/pii/S0164121208001015
- Szyperski, C., Gruntz, D., and Murer, S. (2002). *Component Software: Beyond Object-Oriented Programming.* ACM Press and Addison-Wesley, New York, NY, 2 edition.

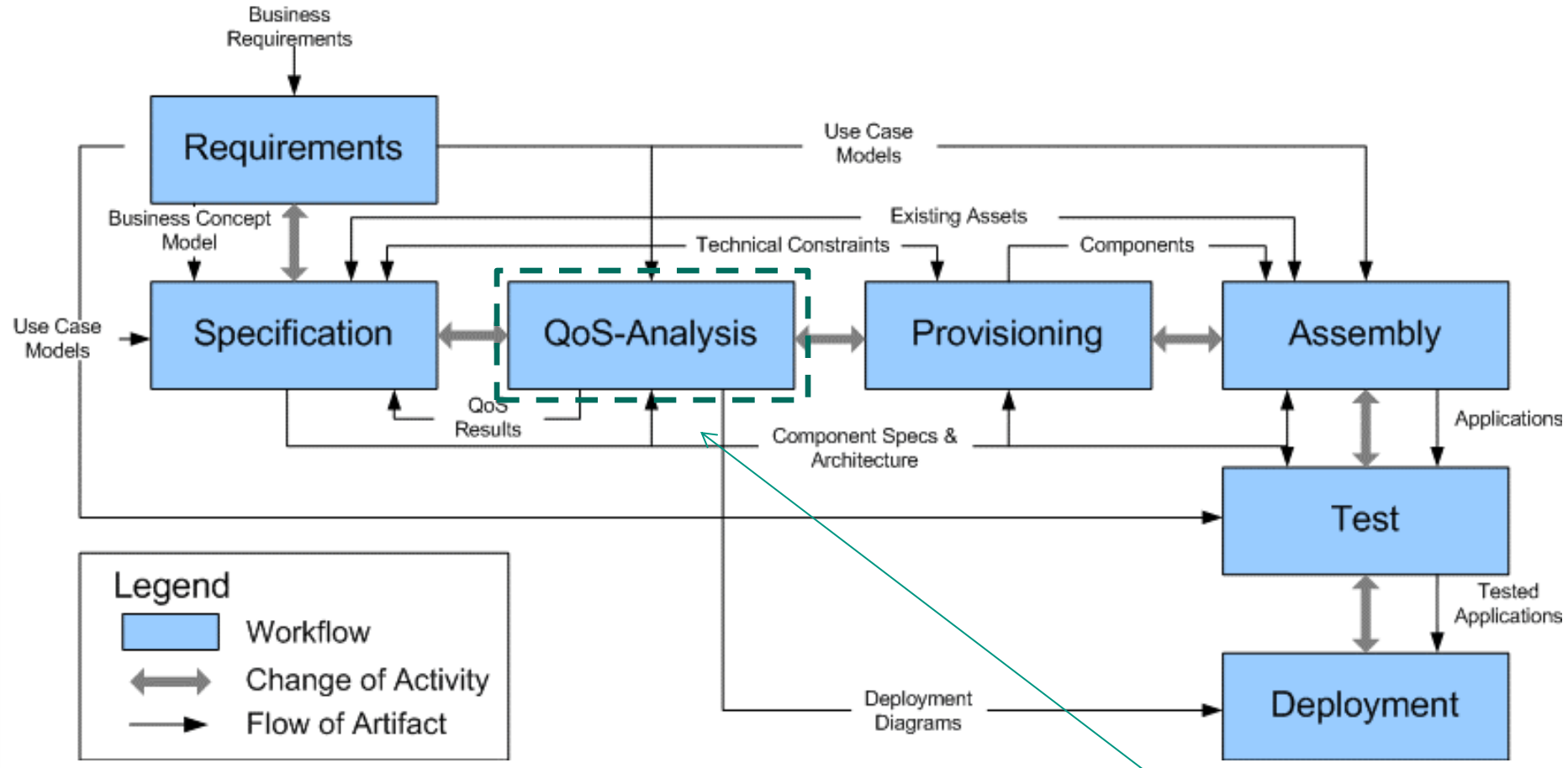


More information and documentation at: http://www.palladio-simulator.com

- Steffen Becker, Palladio Screencast: http://www.palladio-simulator.com/tools/screencasts

[Becker 2008] Steffen Becker. *Coupled Model Transformations for QoS Enabled Component-Based Software Design*, volume 1 of Karlsruhe Series on Software Quality. Universitätsverlag Karlsruhe, 2008.

CBSE Development Process



[Cheeseman2000, Koziolek2006a]