Software Engineering

Natallia Kokash
email: nkokash@liacs.nl

Leiden Institute of Advanced Computer Science

Agenda

- Software reuse
- Component-based system engineering (CBSE)
- Service-oriented architecture (SOA)

How many professional software developers are there worldwide?

Region	Population	Pop/Pro	#Developers
HiTech	2,000,000,000	350	5,700,000
MidTech	2,000,000,000	2,000	1,000,000
LowTech	3,000,000,000	10,000	300,000
Global	7,000,000,000		7,000,000

- \+ hobbyists

How many lines of debugged code does one developer produce in a day's work?

- From 10 to 200
- (sometimes up to 1000)
- 200++ working days per year
- From 2000 to 40000 LOC per developer/year
- 7000000 * 2000 = 14000000000 LOC

Software reuse: six perspectives

Substance:	• defines the essence of reused items
Scope:	• defines the form and extent of reuse
Mode:	• defines how reuse is conducted
Technique:	• defines the approach that is used to implement reuse
Intention:	• defines how elements will be used
Product:	• defines what is reused

Software reuse: examples

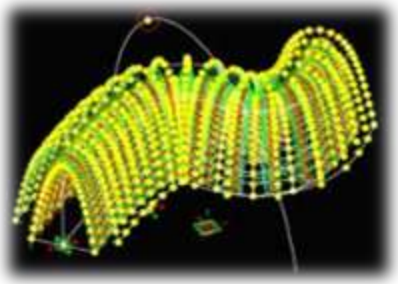
Substance:	• Ideas, components, concepts, …
Scope:	• Horizontal, vertical
Approach:	• Systematic, opportunistic
Technique:	• Compositional, generative
Use:	• black-box, white-box
Product:	• source code, design

Vertical vs. horizontal reuse

- Horizontal domains - parts of a software system are classified according to their functionality:
  - database systems, workflow systems, GUI libraries,…
- Vertical domains - software systems are classified according to the business area:
  - airline reservation systems, medical record systems…
- Horizontal assets are reusable across all (or most) applications
- Vertical assets are *parts of a product line*
  - Microsof Office: opening a document, saving multiple documents, document preview,…
  - Gmail, Orkut & Gtalk: logging in with Google credentials, exchanging messages, persisting chat,…

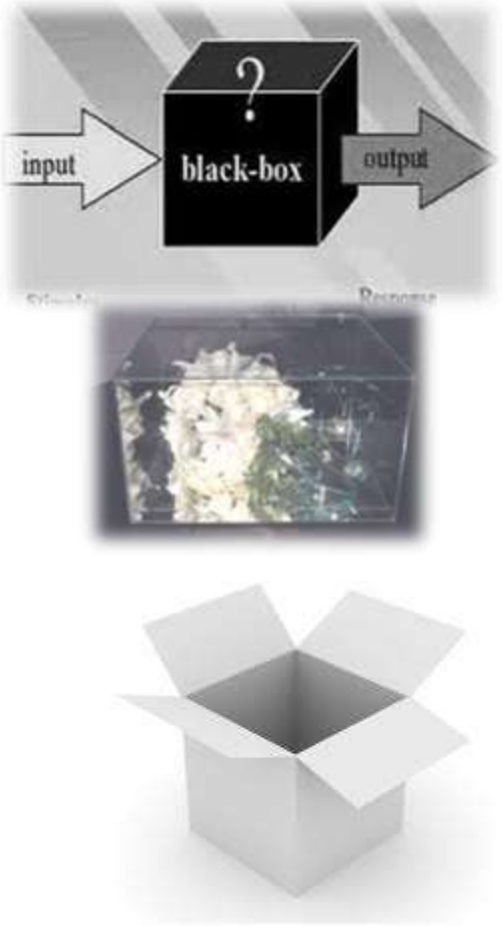
Compositional vs. generative

- Compositional reuse:
  - supports bottom-up development of systems from a repository of available lower-level components;
  - classification and retrieval are important;
  - **Methods**: component composition, code/design scavenging, repositories (e.g., function/class libraries)



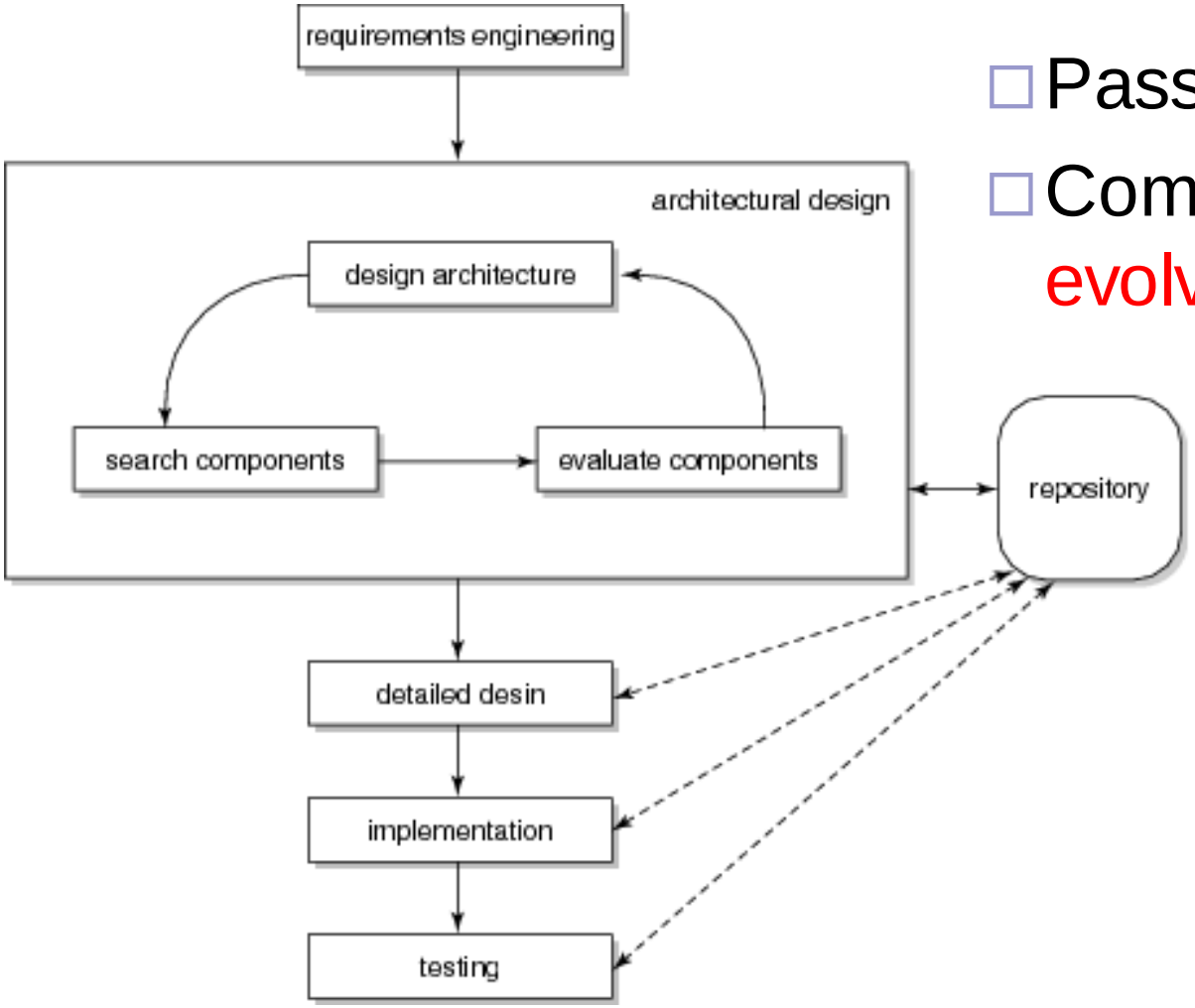
- Generative reuse:
  - often domain-specific;
  - adopting standard system structures (e.g., architectures) and standard interfaces;
  - **Methods**: application generators, language based generators, transformation systems (e.g., parser generation)
- A combined approach is also possible.

Black-box vs. white-box



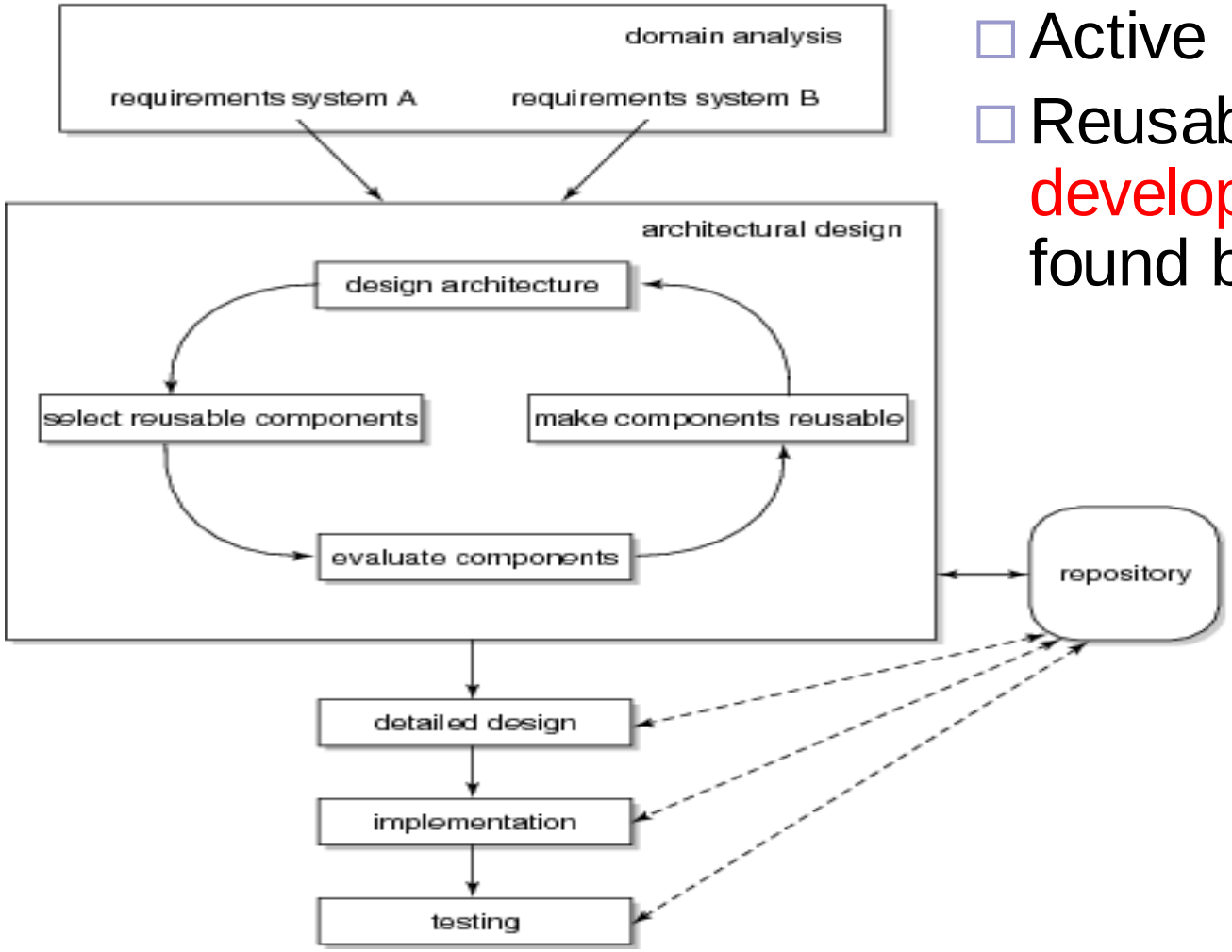
- Black box: only specification is known
- Glass box: internals may be inspected, but not changed
- Grey box: part of the internals may be inspected, limited modification is allowed
- While box: component is open to inspection and modification

Software development with reuse



- Passive
- Component library evolves randomly

Software development for reuse



- Active
- Reusable assets are developed, rather than found by accident

Software development for reuse

- Often two separate development processes:
  - Development of components (involving domain analysis)
  - Development of applications (using the available components)
- **Specific forms:**
  - Component-based software development
  - Software factory
    - Development according to specific, externally-defined end-user requirements through an assembly process
  - Software product lines

Component-based SE (CBSE)

- Increases quality
- Shortens development time

- **Approach:**
  - Search for components
  - Understand/evaluate found components:
    - Quality information
    - Administrative information (developer, modification history)
    - Documentation
    - Interface information
    - Test information
- Adapt components if necessary
- Compose systems from components

A software component:

- Implements some functionality
- Has explicit dependencies through provides and required interfaces
- Communicates through its interfaces only
- Has structure and behavior that conforms to a component model

LEGO analogy

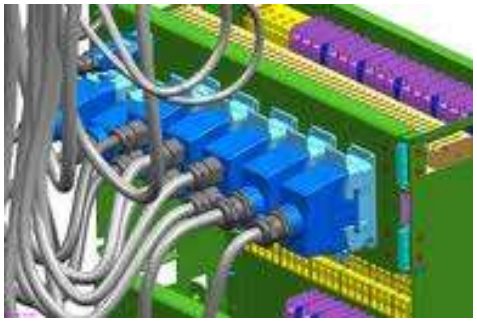
- Set of building blocks in different shapes and colors
- Can be combined in different ways
- Composition through small stubs in one and corresponding holes in another building block

- LEGO blocks are generic and easily composable
- LEGO can be combined with LEGO, not with Meccano

Component model

- Defines the types of building block, and the recipe for putting them together
- Defines standards for:
  - Properties that individual components must satisfy
  - Methods and mechanisms for composing components
- A component has to conform to some component model

Common features of component models



- Infrastructure (instantiation, binding, communication,…)

- Instantiation
- Binding (design time, compile time, …)
- Communication between components
- Discovery of components
- Announcement of component capabilities (interfaces)
- Development support
- Language independence
- Platform independence
- Analysis support
- Support for upgrading and extension
- Support for quality properties

A component technology

- Is the implementation of a component model, by means of:
  - Standards and guidelines for the implementation and execution of software components
  - Executable software that supports the implementation, assembly, deployment, execution of components
- Examples: EJB, COM+, .NET, CORBA

Component technologies

- Common Object Request Broker Architecture (CORBA)
  - Interface Definition Language (IDL)
  - Corba Component Model (CCM)

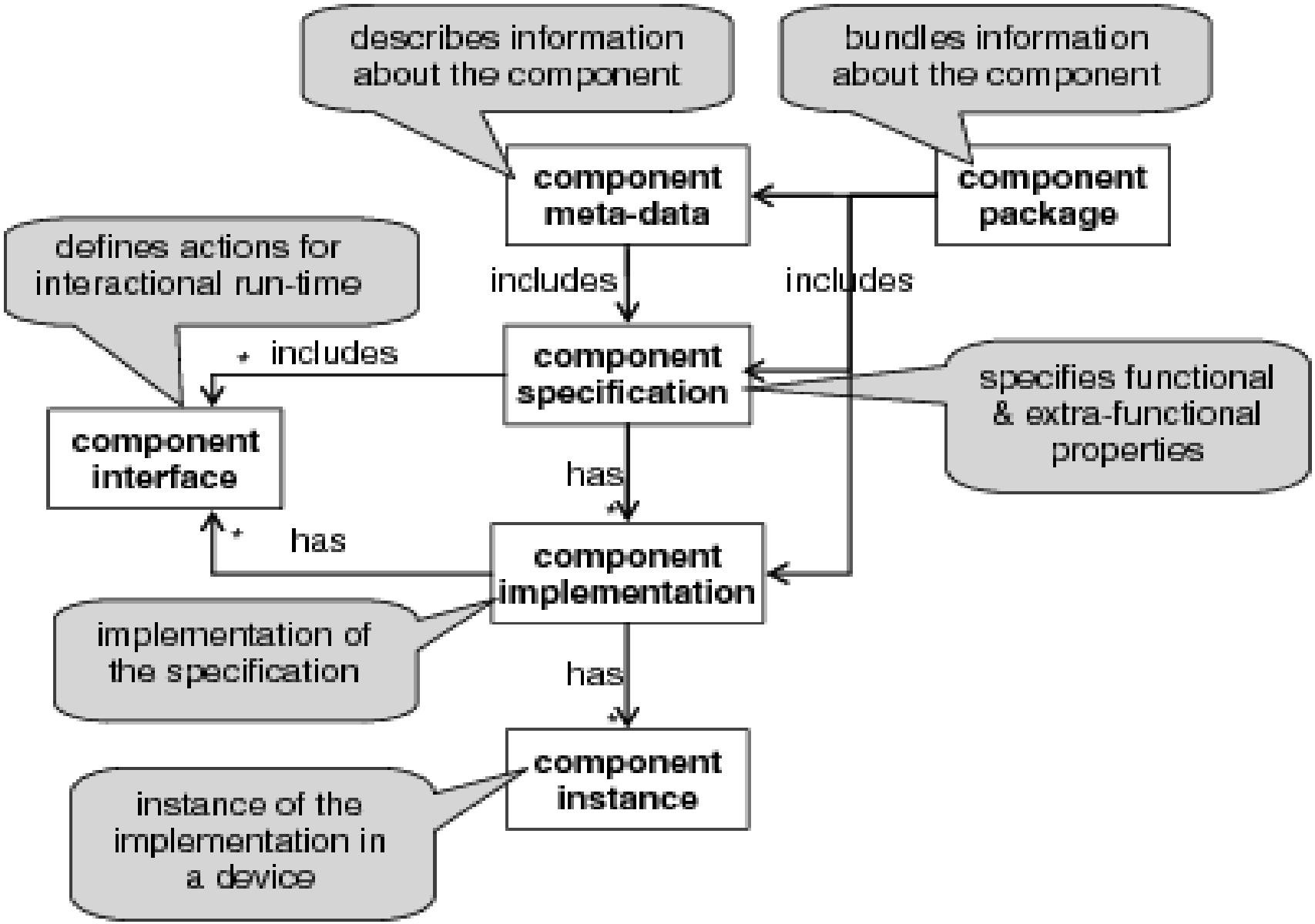
- Enterprise JavaBeans (EJB)
  - Java classes conforming to certain convention (serializable, default constructor, access to properties through Get, Set, Is)
- Component Object Model (COM)
  - Binary-interface standard for language-neutral way of implementing objects
  - OLE, OLE Automation, ActiveX, COM+ and DCOM technologies

- .NET
  - Common Language Runtime (CLR)

Component forms

- Component goes through different stages:

- Across these stages, components are represented in different forms:
  - During development: UML, e.g.
  - When packaging: in a .zip file, e.g.
  - In the execution stage: blocks of code and data


describes information about the component
bundles information about the component
component meta-data
component package
defines actions for interactional run-time
includes
includes
includes
component specification
specifies functional & extra-functional properties
component interface
has
has
component implementation
implementation of the specification
has
component instance
instance of the implementation in a device

Component specification vs. component interface

- Specification describes properties to be realized: realization contract
- Interface describes how components interact: usage contract

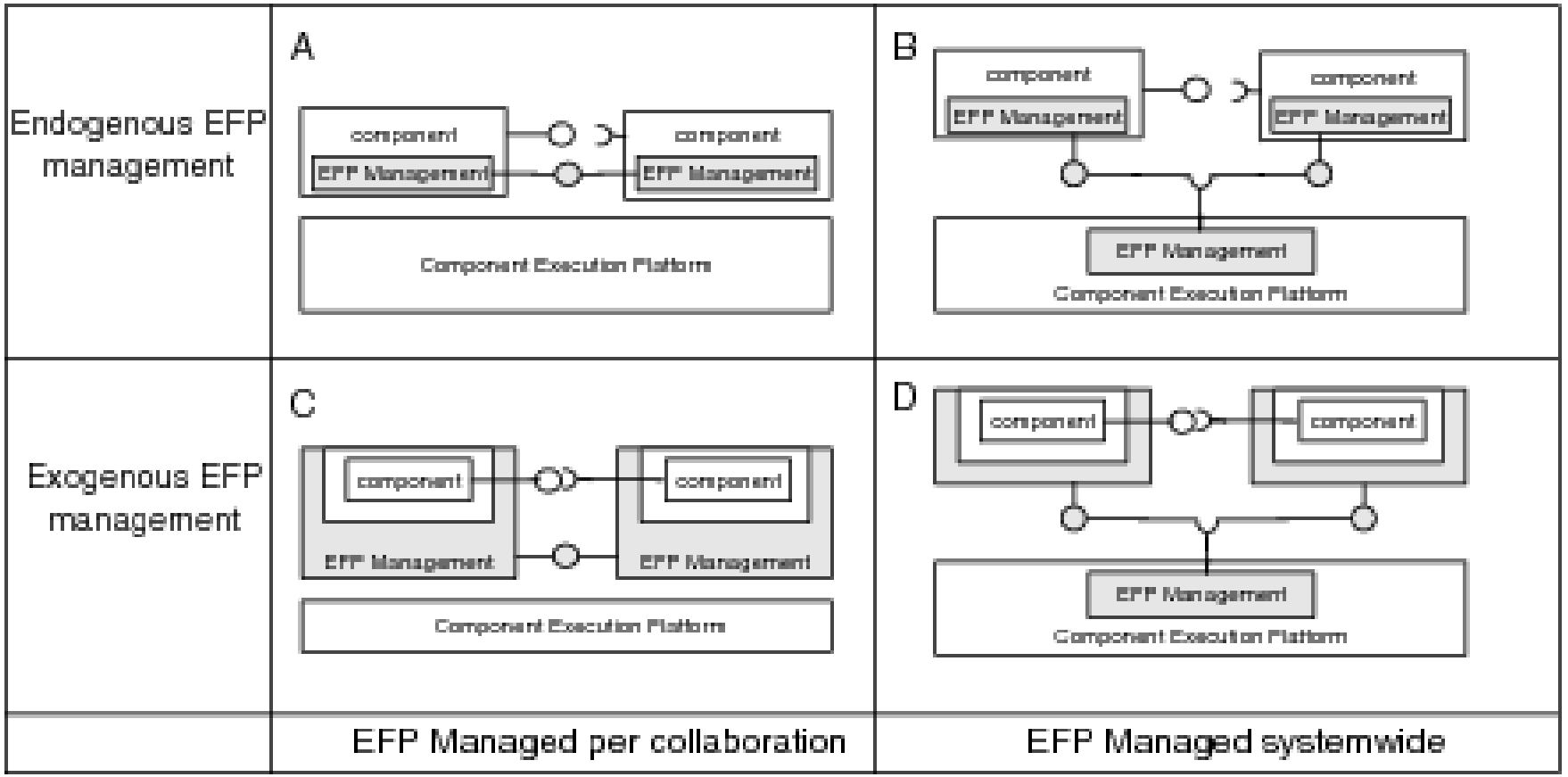
- Different in scope:
  - specification is about the component as a whole
  - an interface might be about part of a component only

Managing quality in CBSE

Who manages the quality:	• component, execution platform
Scope:	• per-collaboration, system-wide

- **Approaches:**
  - Endogenous:
    - Managed per collaboration: left to the designers (e.g. when integrating COTS components) (**A**)
    - Managed systemwide: model provides standardized facilities for managing qualities (**B**)
  - Exogenous:
    - Managed per collaboration: component only addresses functionality, quality in surrounding container (**C**)
    - Managed systemwide: similar to C, but container interacts with platform on quality issues (**D**)

Quality management in CBSE



CBSE system development process

Requirements:	• also considers *availability* of components
Analysis and design:	• very similar to what we normally do
Implementation:	• less coding, focus on selection of components, provision of glue code
Integration:	• largely automated
Testing:	• verification of components is necessary
Release:	• as in classical approaches
Maintenance:	• replace components

Languages to describe component compositions

- Module Interconnection Languages (MIL)
  - Describe the uses relationships among modules
  - Static type checking, version control, C++ like syntax (but not a programming language)
- Architecture Description Languages (ADL)
  - Focus on conceptual architecture and explicit treatment of connectors
  - See lecture 4 (Software Architecture)

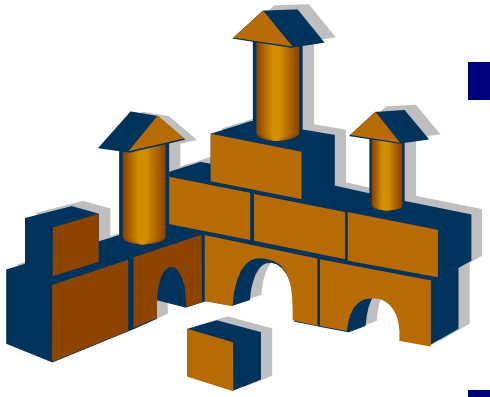
Component development process

Requirements:	• combination of top-down (from system) and bottom-up (generality)
Analysis and design:	• generality is an issue, assumptions about system (use) must be made
Implementation:	• largely determined by component technology
Testing:	• extensive (no assumptions of usage!), and well-documented
Release:	• not only executables, also metadata

Component maintenance

- Who is responsible: producer or consumer?
- Blame analysis: relation between manifestation of a fault and its cause, e.g.
  - Component A requires more CPU time
  - As a consequence, B does not complete in time as required by C, so
  - C issues a time-out error to its user
  - Analysis: goes from C to B to A to input of A
  - Who does the analysis, if producers of A,B,C are different?

Architecture and CBSE



- Architecture-driven: top-down: components are identified as part of an architectural design
- Product line: family of similar products, with 1 architecture
- COTS-based: bottom-up, architecture is secondary to components found

Evolution of programming paradigms

- Imperative computing (1950-1970s). FORTRAN was the flagship language of this approach
- Procedural computing (1970-1980s), emphasizes modular software design. Programming languages: Algol, Pascal, and C.
- Object-Oriented Computing (1980s-…), emphasizes abstraction and code reuse. Programming languages: C++, Java, and C#.
- Service-Oriented Computing (late 1990s-…). standardization of service interfaces for platform-independent pervasive code reuse. *Service discovery* is vital to SOC. Programming languages: Java and C#.

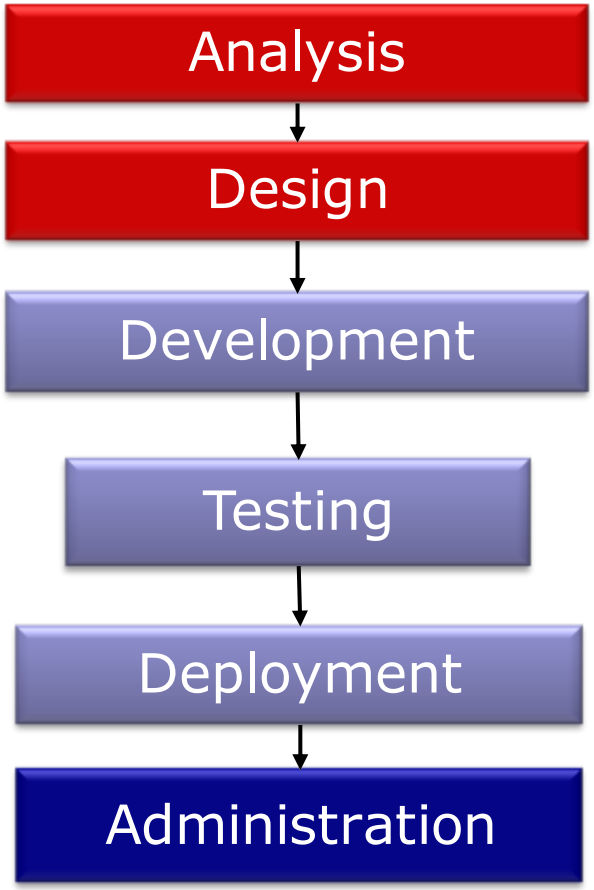
Service-oriented computing

- Services are entities that provide some capability to its clients (by exchanging messages)
- Service description, service communication
- Service-Oriented Architecture (SOA)
- Service-Oriented Software Engineering (SOSE)
- Web service technology

Main characteristics of services

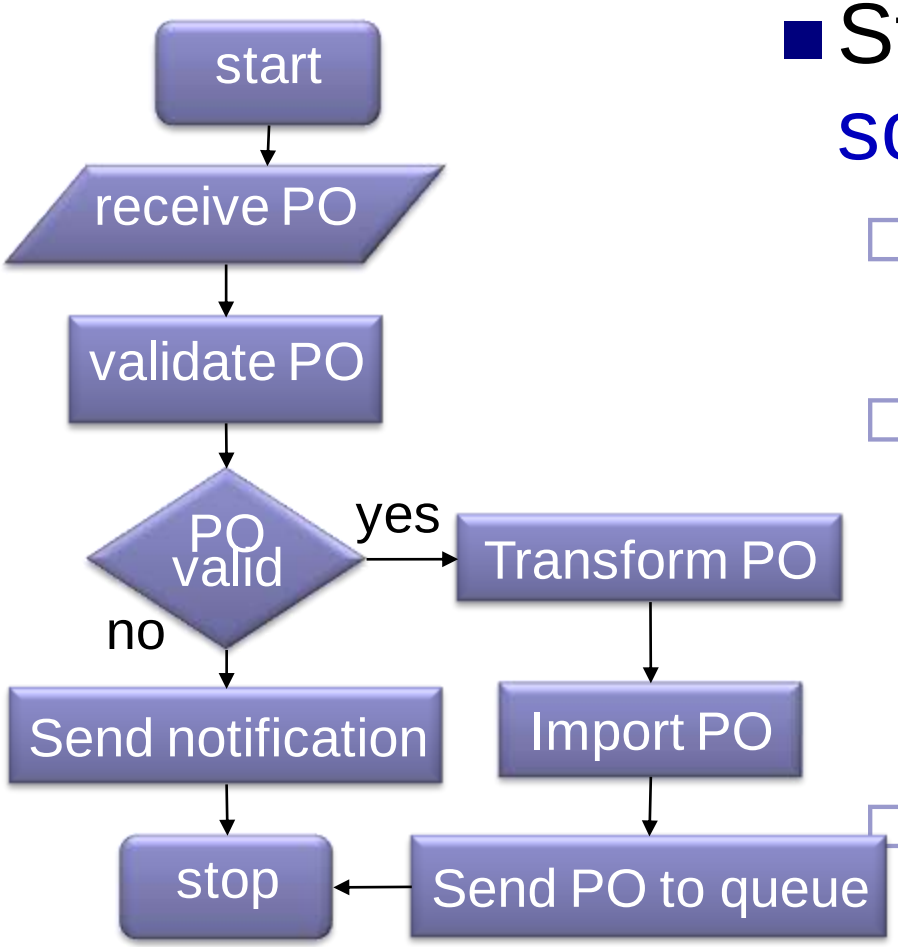
- Can be discovered
- Can be composed to form larger services
- Adhere to a service contract
- Loosely coupled
- Stateless
- Autonomous
- Hide their logic
- Reusable
- Use open standards
- Facilitate interoperability

SOSE



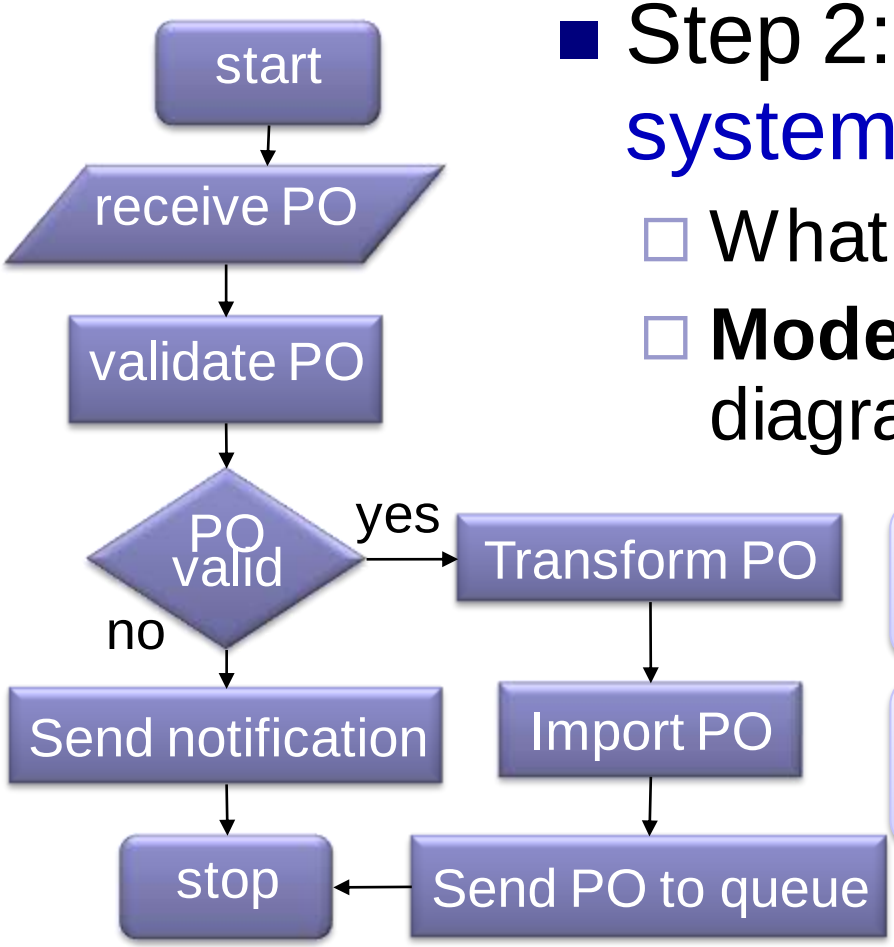
- Development of software systems by composition of reusable services
- Shares many characteristics with CBSE (+ locate services at run-time)
- **Approaches:**
  - Top-down
  - Bottom-up
  - Agile

Service-oriented analysis



- Step 1: define analysis scope
  - Mature and understood business requirements
  - Can lead to:
    - process-agnostic services (generic service portfolio)
    - services delivering business-specific tasks
  - **Models**: UML use case or activity diagrams

Service-oriented analysis

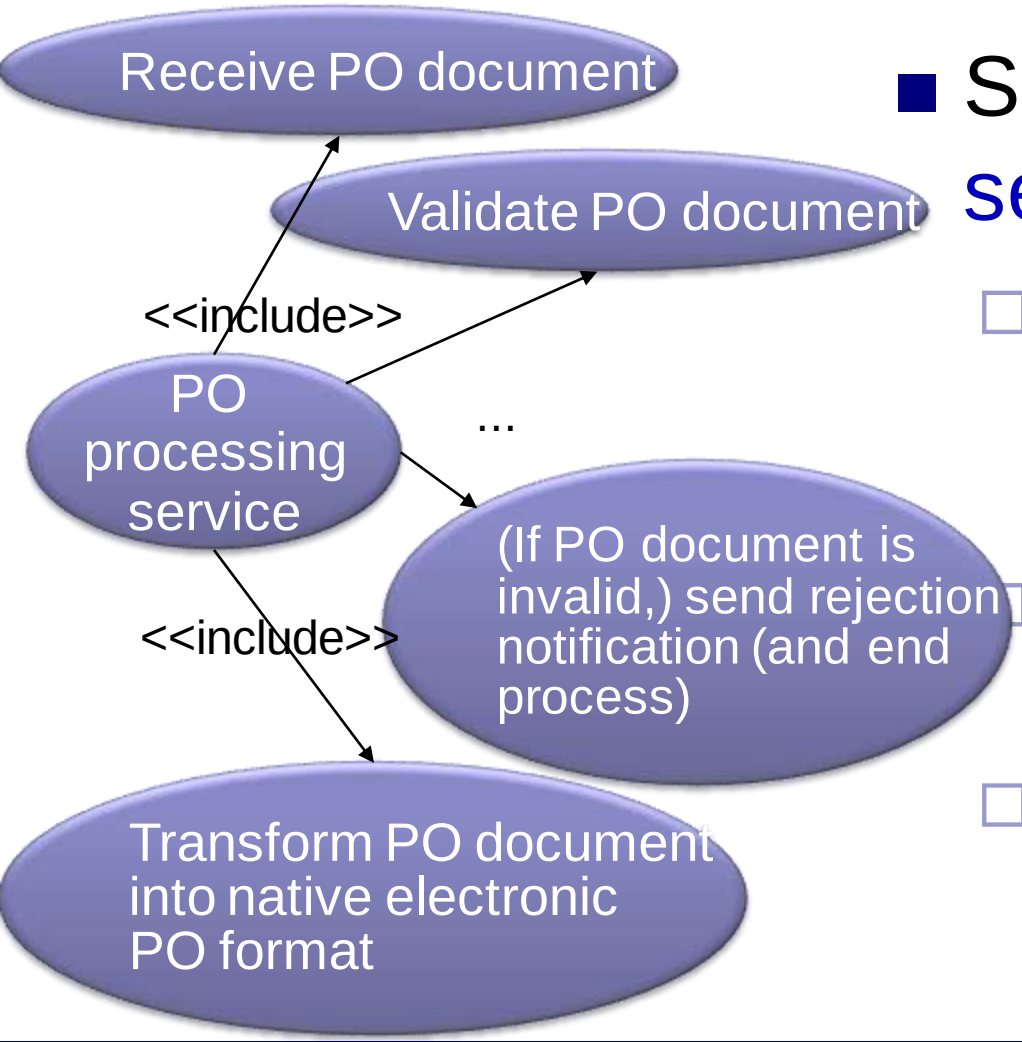


- Step 2: Identify automation systems
  - What is already implemented?
  - **Models**: UML deployment diagram, mapping tables

(XML -> native format)
currently custom component
service candidate

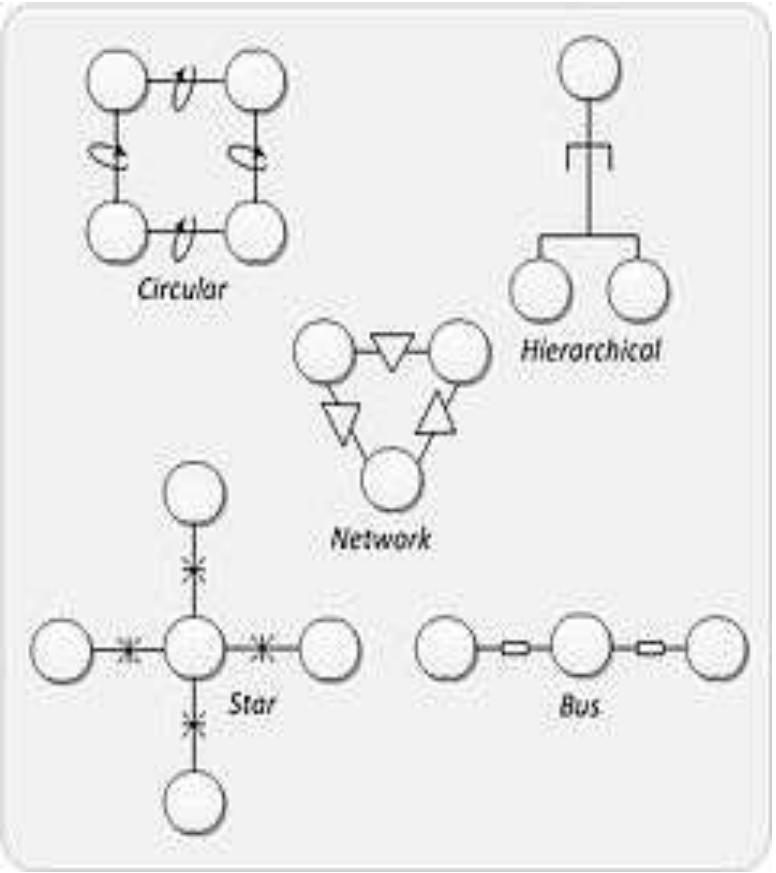
into accounting system
currently custom legacy
service candidate

Service-oriented analysis



- Step 3: Model candidate services
  - Create conceptual models of service candidates
  - How to compose services?
  - **Models**: BPM, UML use case or class diagrams

Service-oriented modeling styles



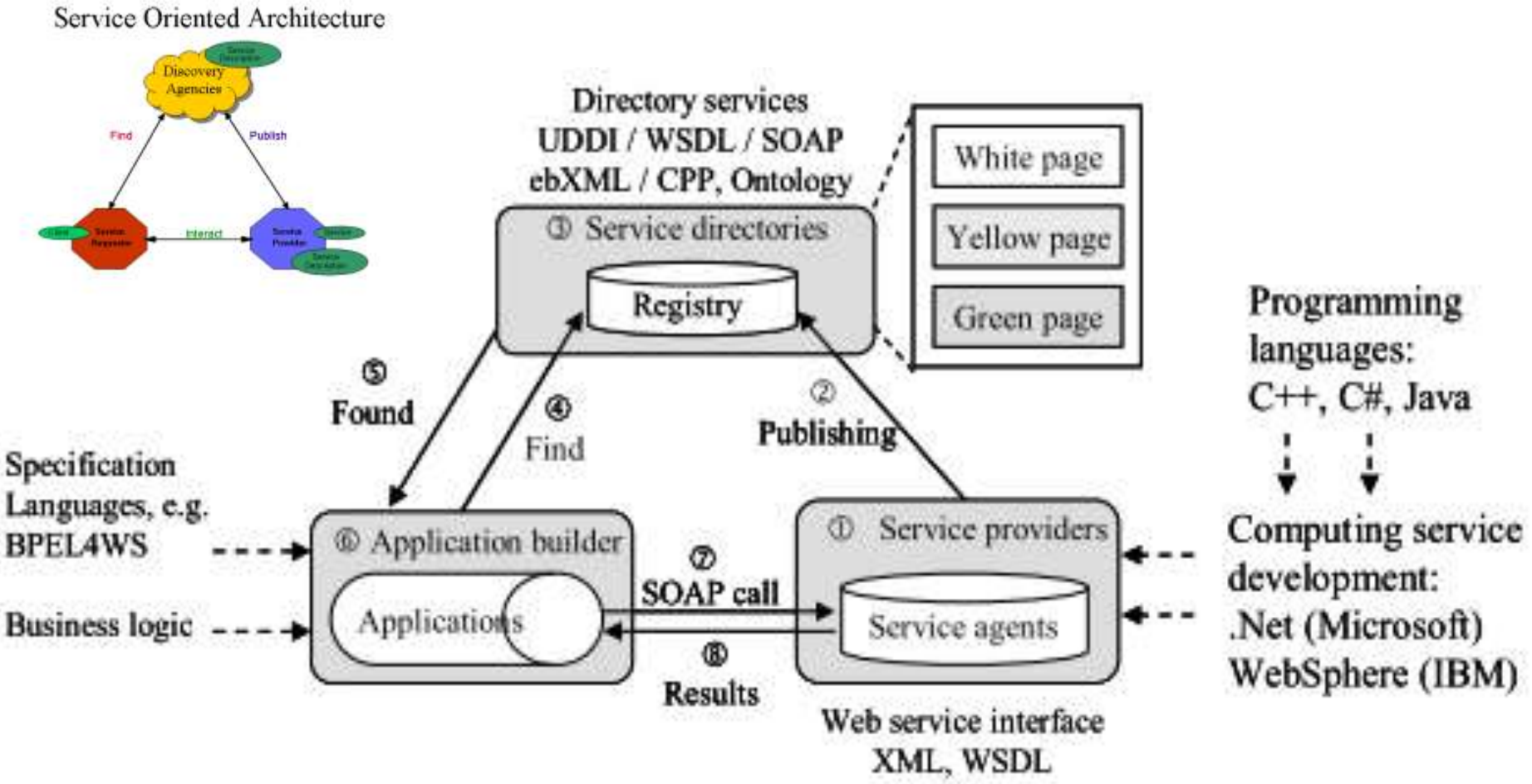
- Circular: message exchange in a circular fashion
- Hierarchical: certain message exchange routes between consumers and services.
- Network: establishes “many to many” relationship between services, their peer services, and consumers.
- Star: the central service passes messages to its extending arms
- Bus: introduces an intermediary service to connect consumers with service providers

Web services

- XML-based data sharing
- **Description:** Web Service Description Language (WSDL)
- **Messages**: Simple Object Access Protocol (SOAP)
- **Distribution**: Universal Description Discovery and Integration (UDDI)
- **Composition (coordination):** Business Process Execution Language (BPEL)

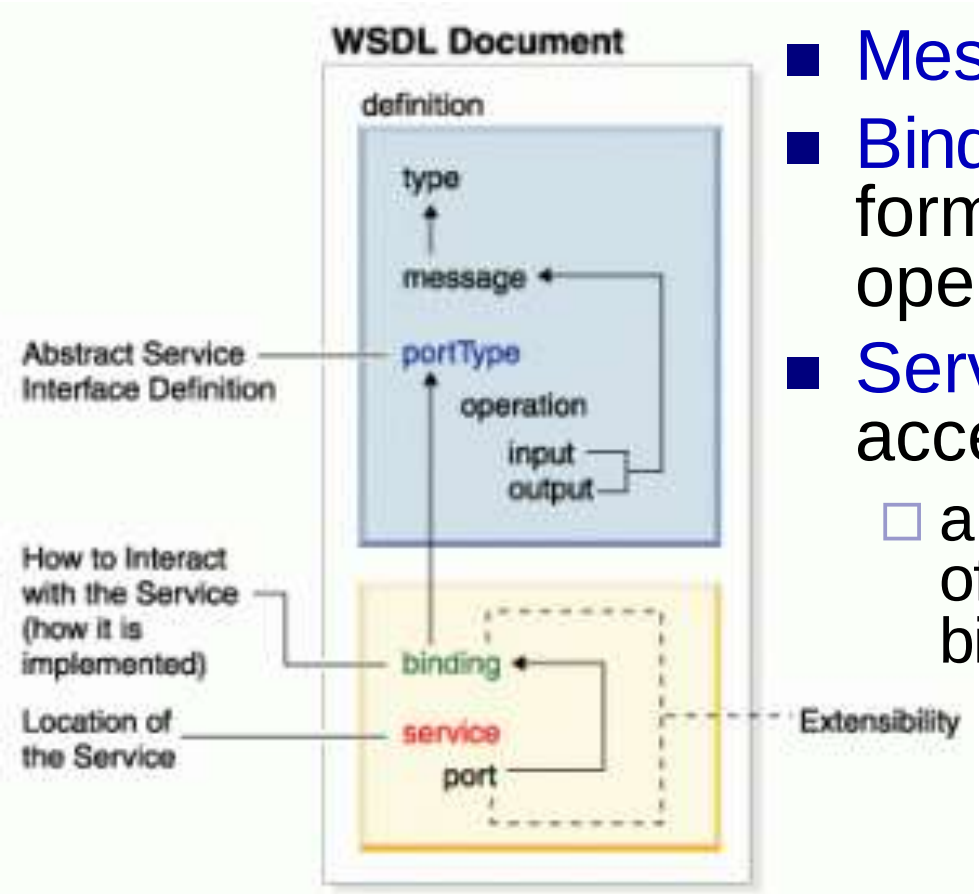
- Also:
  - Secure communication: WS-Security
  - Quality of Service provision: WS-Policy,…

Web service technology



WSDL

- Data types (using XML Schema)
- Interfaces (`portType`)
- Messages (`Messages`)
- Bindings define transport and format details for the operations and messages
- Services (end points for accessing a service):
  - a service can have several ports of the same type with different bindings



XML schema language

- Allows the content of an element or attribute to be validated against a data type.
- Provides a set of 19 primitive data types (e.g., anyURI, boolean, date, string)
- Allows new data types to be constructed from these primitives by three mechanisms:
  - restriction (reducing the set of permitted values),
  - list (allowing a sequence of values), and
  - union (allowing a choice of values from several types).

WSDL example: Hello service

```
<definitions name="HelloService"
   targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>
   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>

   <portType name="Hello_PortType">
      <operation name="sayHello">
         <input message="tns:SayHelloRequest"/>
         <output message="tns:SayHelloResponse"/>
      </operation>
   </portType>
```

```
   <binding name="Hello_Binding" type="tns:Hello_PortType">
   <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="sayHello">
      <soap:operation soapAction="sayHello"/>
      <input>
         <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:examples:helloservice"
            use="encoded"/>
      </input>
      <output>
         <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:examples:helloservice"
            use="encoded"/>
      </output>
   </operation>
   </binding>

   <service name="Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding="tns:Hello_Binding" name="Hello_Port">
         <soap:address
            location="http://www.examples.com/SayHello/">
      </port>
   </service>
</definitions>
```

Analysis of the example

- **`Definition`**: HelloService
- **`Type`**: Using built-in data types and they are defined in XMLSchema.
- **`Message`**:
  - sayHelloRequest : firstName parameter
  - sayHelloresponse: greeting return value
- **`Port Type`**: sayHello **operation** that consists of a request and response service.
- **`Binding`**: Direction to use the SOAP HTTP transport protocol.
- **`Service`**: Service available at http://www.examples.com/SayHello/ .
- **`Port`**: Associates the binding with the URI http://www.examples.com/SayHello/ where the running service can be accessed.

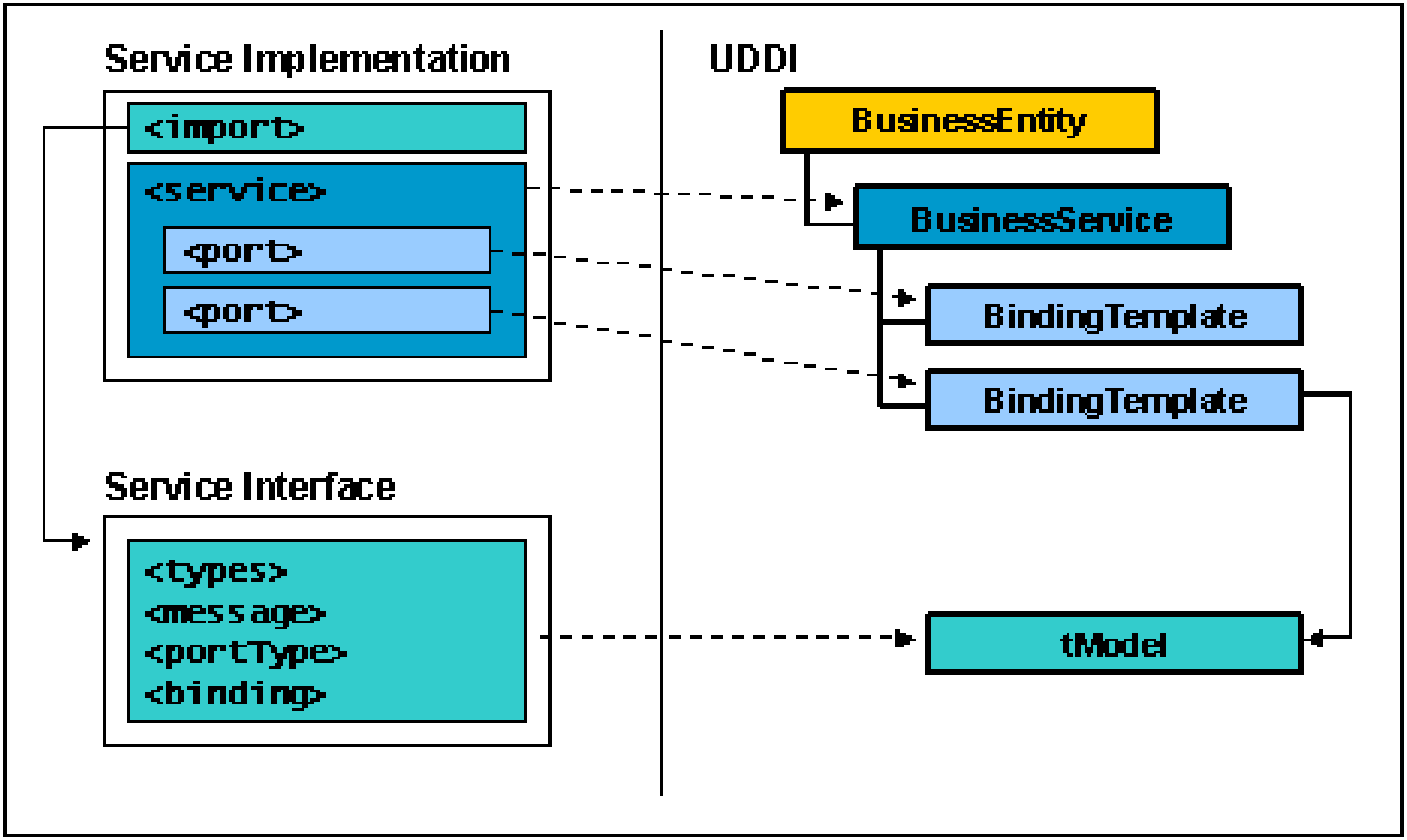
UDDI

- XML-based registry for businesses worldwide to list themselves:
  - White pages – address, contact, and known identifiers;
  - Yellow pages – industrial categorizations based on standard taxonomies;
  - Green pages — technical information about services exposed by the business.
- A mechanism to register and locate web services
- Public vs. private regsitries

UDDI registry entry

- `businessEntity` – information about the organization: name, description, a unique identifier, etc.
- `businessServices` – information about actual provided services
- `bindingTemplates` – technical information to link services to implementation information (e.g., pointer to a web site or service description)

UDDI registry entry



UDDI tModel

```
<?xml version="1.0"?>
<tModel tModelKey="">
  <name>http://www.getquote.com/StockQuoteService-interface</name>

  <description xml:lang="en">
    Standard service interface definition for a stock quote service.
  </description>

  <overviewDoc>
    <description xml:lang="en">
      WSDL Service Interface Document
    </description>
    <overviewURL>
      http://www.getquote.com/services/
SQS-interface.wsdl#SingleSymbolBinding
    </overviewURL>
  </overviewDoc>

  <categoryBag>
    <keyedReference tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
                    keyName="uddi-org:types" keyValue="wsdlSpec"/>
    <keyedReference tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
                    keyName="Stock market trading services"
                    keyValue="84121801"/>
  </categoryBag>
</tModel>
```

SOAP

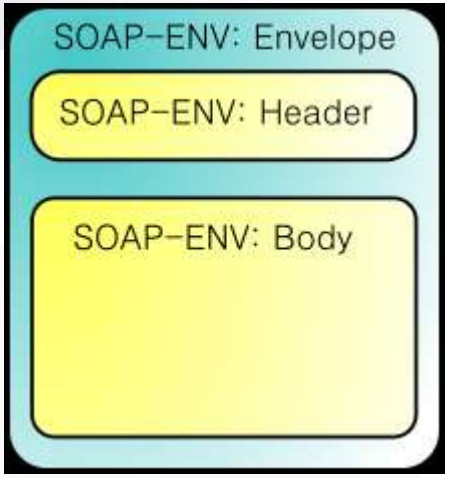
- A protocol specification for exchanging structured (XML-based) information
- Relies on application layer protocols:
  - Hypertext Transfer Protocol (HTTP)
  - Simple Mail Transfer Protocol (SMTP)

- **Characteristics:**
  - Extensibility (WS-security and WS-routing are among the extensions under development),
  - Neutrality (SOAP can be used over any transport protocol)
  - Independence (SOAP allows for any programming model)

SOAP

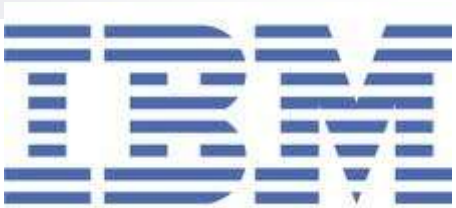
- Processing model - defines the rules for processing a SOAP message
- Extensibility model - defines the concepts of SOAP features and SOAP modules
- Underlying protocol binding – describes the rules for defining a binding to an underlying protocol that can be used for exchanging SOAP messages between SOAP *nodes (e.g., sender, receiver, intermediary)*
- Message construct - defines the structure of a SOAP message

SOAP




```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

BPEL



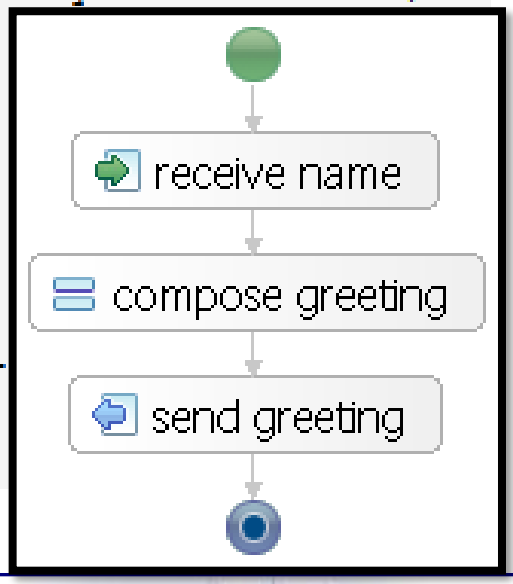
- OASIS standard (Organization for the Advancement of Structured Information Standards)
- Executable language for specifying actions within business processes with web services
- Used to model the behavior of both executable and abstract processes
- **Orchestration vs. choreography:**
  - Orchestration - specifies an executable process that involves message exchanges with other systems (central control, by conductor)
  - Choreography - specifies a protocol for peer-to-peer interactions (distributed system, dancing team)
- BPEL is an orchestration language

BPEL provides

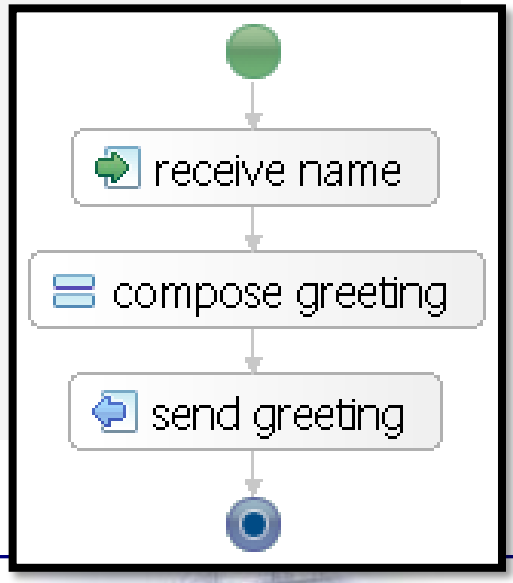
- Facilities to enable sending and receiving messages
- A property-based message correlation mechanism
- XML and WSDL typed variables
- An extensible plug-in model to allow writing expressions and queries
- Structured-programming constructs:
  - *if-then-elseif-else, while, sequence* (commands in order) and *flow* (commands in parallel)
- A scoping system to allow the encapsulation of logic with *local variables*, *fault-handlers*, *compensation-handlers* and *event-handlers*
- Control concurrent access to variables

BPEL example: HelloWorld

```
<process name="HelloWorld" targetNamespace="http://jbpm.org/examples/hello"
  xmlns:tns="http://jbpm.org/examples/hello"
  xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <partnerLinks>
    <!-- establishes the relationship with the caller agent -->
    <partnerLink name="caller" partnerLinkType="tns:Greeter-Caller" myRole="Greeter" />
  </partnerLinks>

  <variables>
    <!-- holds the incoming message -->
    <variable name="request" messageType="tns:nameMessage" />
    <!-- holds the outgoing message -->
    <variable name="response" messageType="tns:greetingMessage" />
  </variables>
```



```
<sequence name="MainSeq">

  <!-- receive the name of a person -->
  <receive name="ReceiveName" operation="sayHello" partnerLink="caller"
    portType="tns:Greeter" variable="request" createInstance="yes" />

  <!-- compose a greeting phrase -->
  <assign name="ComposeGreeting">
    <copy>
      <from expression="concat('Hello, ', bpel:getVariableData('request', 'name'), '!')" />
      <to variable="response" part="greeting" />
    </copy>
  </assign>

  <!-- send greeting back to caller -->
  <reply name="SendGreeting" operation="sayHello" partnerLink="caller"
    portType="tns:Greeter" variable="response" />

</sequence>

</process>
```

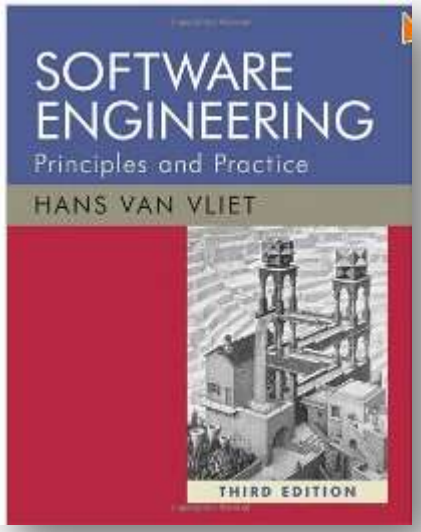


Summary

- Software reuse:
  - We can reuse different things: code, design, …
  - Reuse can be *systematic* or *opportunistic*
- Component-based SE
  - Various component models exist
- Service-Oriented Computing:
  - SOA is an architectural style
  - Most important characteristic: dynamic discovery of services

Homework

- Read chapters 17-19
- Try to develop/reuse components/services for your assignment (using different technologies e.g., .NET, Web Services)



- …
- 12. Software Design
- 13. Software Testing
- 14. Software Maintenance
- 17. Software Reusability
- 18. Component-Based Software Engineering
- 19. Service Orientation
- 20. Global Software Development