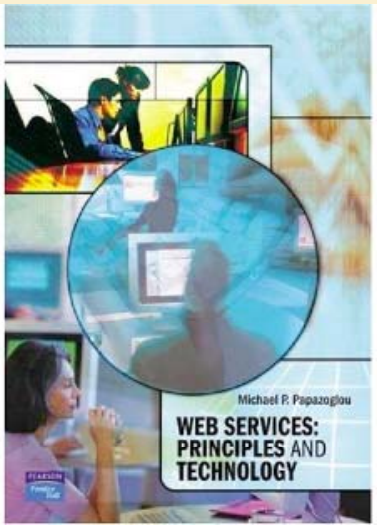
Web Services: Principles & Technology

# Chapter 15
# Web Services Development Lifecycle

Mike P. Papazoglou
mikep@uvt.nl




UNIVERSITEIT VAN TILBURG





## Topics

- *Web services development*
- *Properties of service development methodology*
- *Qualities of service development methodology*
- *Web services development lifecycle*
- *Service analysis*
- *Service design*
- *Service construction*
- *Service provisioning*
- *Service deployment*

# Web services development methodology

- Introducing a thin SOAP/WSDL/UDDI veneer atop existing applications or components that implement Web services is by now widely practiced by the software industry.
  - Unless the nature of the component makes it suitable for use as a Web service, and most are not, it takes serious thought and redesign effort to properly deliver component functionality through a Web service.
- A methodology is of critical importance to specify, construct, refine, and customize highly volatile business processes from internally and externally available Web services.
  - A sound methodology allows an organization to avoid the pitfalls of deploying an uncontrolled maze of services and provide a solid foundation for service enablement in an orderly fashion so that Web services can be efficiently used in SOA-based business applications.





# Related development methodologies

Modeling techniques such as object-oriented analysis and design (OOAD), component-based development (CBD), and business process modeling (BPM) although useful within their own scope, cannot be directly applied to service-oriented development.

- OOAD allows designers to focus attention on units such as classes and objects, which match enterprise or business concepts.
  - its level of granularity is focused on the class level, which resides at too a low level of abstraction for business service modeling and creates tight coupling.
- CBD offers a “separation of internal and external perspectives;” however, is limited in approaching flexibility, composability, and reusability.
  - The selection of a service is usually done dynamically on the basis of a set of policies. Use of installed components does not allow for the same kind of reuse and dynamic behavior.
  - CBD carries with it all the difficulties of object modeling and multiplies the complexity by increasing the scale of the model. Let alone if models are extended across enterprise boundaries.

# Related development methodologies (continued)

- BPM (Business process modeling) examines business processes, identifies ways to improve them, and addresses barriers that impede their ability to achieve their business goals.
  - The result is a practical action plan that provides a blueprint for achieving an organization's goals and implementing change, while respecting the enterprise's strategic mission.





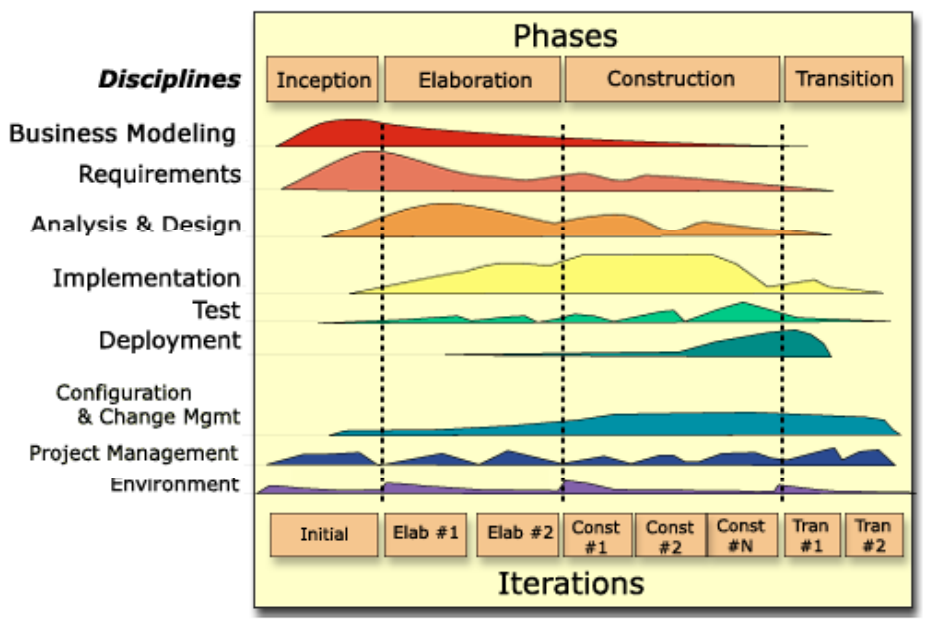
# Shortcomings

- OOAD, CBD and BPM are not grounded on SOA principles, fail to address the three key elements of an SOA: services, service assemblies (composition), and components realizing services.

- They also do not support:
  - distributed and hybrid service development and deployment;
  - service provisioning;
  - service management.

- OOAD, CBD, and BPM only address part of the requirements of service-oriented computing applications. These practices *fail* when they attempt to develop service-oriented solutions *while being applied independently of each other*.

# Industry practices: rational unified process (RUP)





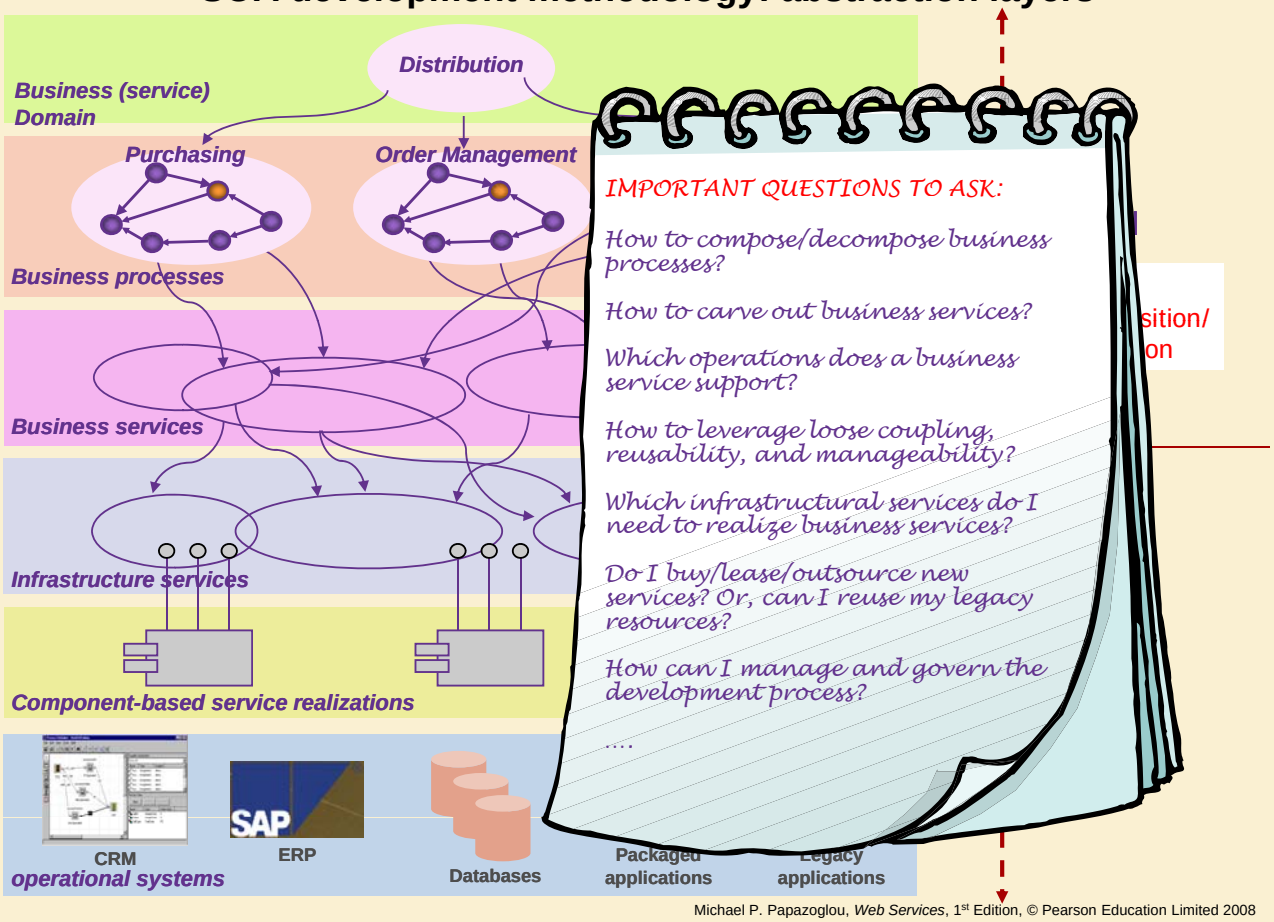



# Topics

- *Web services development*
- *Properties of service development methodology*
- *Qualities of service development methodology*
- *Web services development lifecycle*
- *Service analysis*
- *Service design*
- *Service construction*
- *Service provisioning*
- *Service deployment*

Slide 15.9

## SOA development methodology: abstraction layers






Slide 15.10

## Important milestones

- *Reuse existing functionality*
- *Minimize costs of disruption*
- *Employ an incremental mode of integration*
- *Provide increased flexibility*
- *Provide scalability*
- *Provide (design for) compliance with standards*

# Service-oriented design and development principles

- Service coupling
- Service cohesion
- Service granularity





# Service coupling

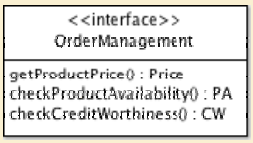
Coupling is the degree of interdependence between any two business processes.

- *Representational service coupling*: Services should not depend on specific representational or implementation details and assumptions of one another. Leads to:
  - Service interchangeability
  - Service versioning
- *Identity service coupling:* Connection channels between services should be unaware of who is providing the service.
- *Communication protocol coupling:* A sender of a message should rely only on those effects necessary to achieve effective communication.

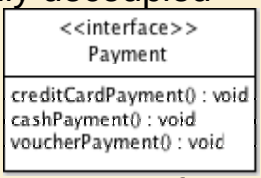
# Service cohesion

Cohesion is the degree of the strength of functional relatedness of operations within a service.

- *Functional service cohesion:* Performs one and only one problem-related task and contain only services necessary for that purpose.




- *Communicational service cohesion:* Activities and services use the same input and output messages. Leads to cleanly decoupled business processes.




- *Logical service cohesion:* Services all contribute to tasks of the same general category. They perform a set of independent but logically similar functions (alternatives).





# Service granularity

- ***Service granularity*** is the scope of functionality exposed by a service.
- Services may come at two levels of granularity:
  - A coarse-grained interface might be the complete processing for a given service, e.g., “SubmitPurchaseOrder”
    - the message contains all business information needed to define a PO
    - A fine-grained interface might have separate operations for “CreateNewPurchase Order,” “SetShippingAddress,” “AddItem,” etc.
- Fine-grained services usually provide basic data access or rudimentary operations.
- Coarse-grained services are composed from finer-grained services.

# Service granularity concerns

- A service interface (WSDL port type) should generally contain more than one operation, supporting a particular business activity


- The frequency of message exchange is an important factor. Sending and receiving more information in a single request is more efficient than sending many fine-grained messages.
  - *Several redundant, fine-grained services lead to increased message traffic, and tremendous overhead, and inefficiency.*
  - *A small collection of coarser-grained services – each of which implements a complete business process – that are usable in multiple scenarios is a better option.*

- Heuristics identify the right level of granularity for services.

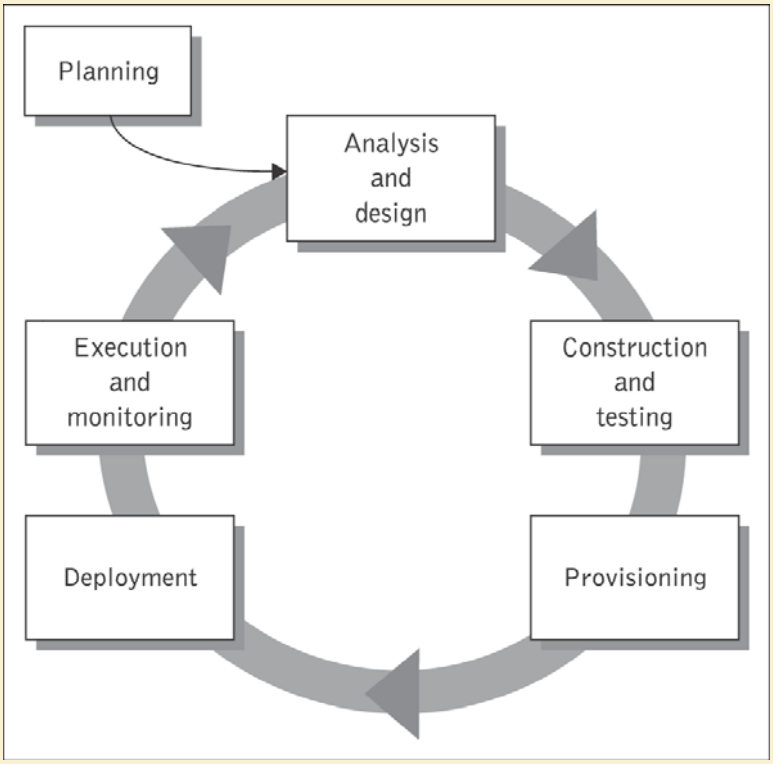




# Topics

- *Web services development*
- *Properties of service development methodology*
- *Qualities of service development methodology*
- *Web services development lifecycle*
- *Service analysis*
- *Service design*
- *Service construction*
- *Service provisioning*
- *Service deployment*

# Web services development lifecycle








# Service oriented design and development concerns

- Manage the entire services lifecycle
- Establish a platform, programming model, and managing services
- Adopt “best-practices” and tools
- Deliver high-quality SOA solutions

# Topics

- *Web services development*
- *Properties of service development methodology*
- *Qualities of service development methodology*
- *Web services development lifecycle*
- *Service analysis*
- *Service design*
- *Service construction*
- *Service provisioning*
- *Service deployment*

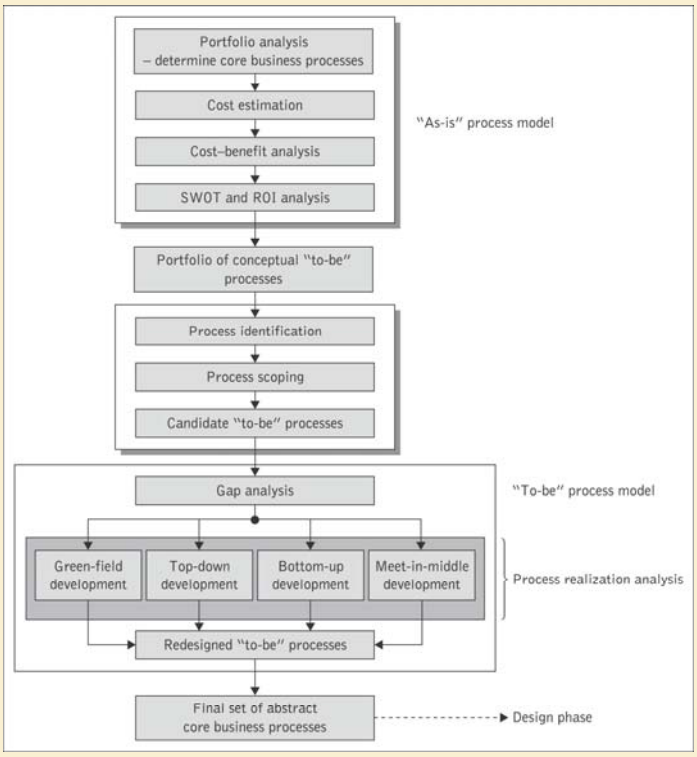




# Service analysis

- Service analysis aims at identifying and describing the processes in a business problem domain, and discovering potential overlaps and discrepancies between processes under construction and available system resources needed to realize services.
- It helps prioritize business processes where SOA can contribute to improvements and offer business value potential.
- It identifies, conceptualizes, and rationalizes business processes as a set of interacting services.
- Develops in-depth understanding of functionality, scope, reuse, and granularity of candidate business processes and services.

# Service analysis steps





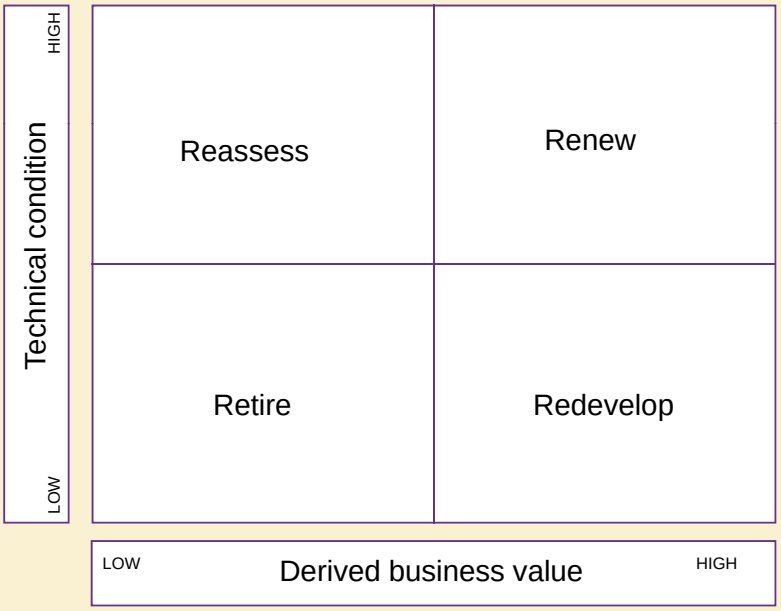



# Portfolio analysis

- Portfolio analysis requires that applications and business processes that are candidates for re-engineering be prioritized according to their technical quality and business value.
  - Business objectives are derived from business strategies and processes, while technical assets are mapped and weighted against these business objectives and evaluated using a comprehensive set of metrics.

- Current appliations and processes are assessed using various techniques including:
  - Reverse-engineering
  - Architectural Knowledge Elicitation
  - Business Process Mapping
  - Process Mining.

# Portfolio analysis (continued)





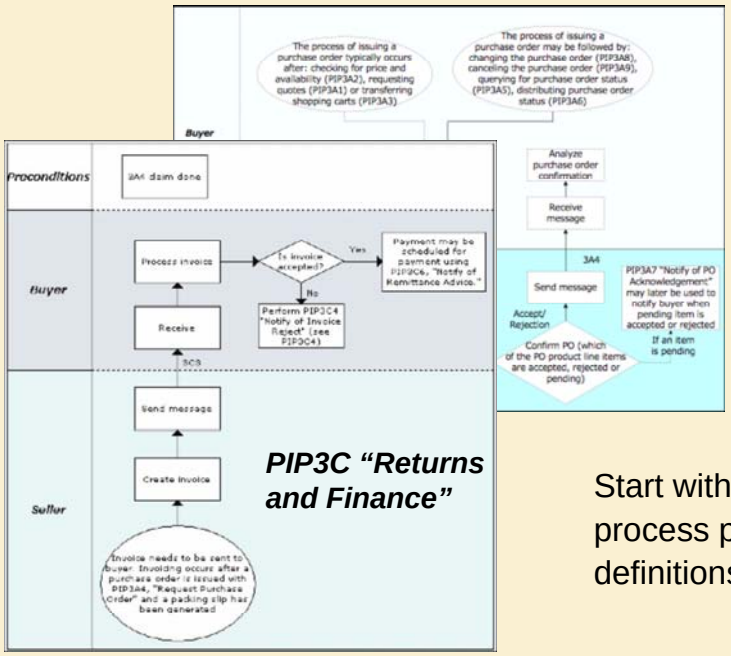



# Service identification and scoping

- Service identification inspects enterprise core business entities, ascertains business concepts and leads to the formation of conceptual business processes and business services. Takes into account important issues such as:
  - consolidation;
  - decomposition;
  - reuse simplification; and
  - refactoring of legacy assets.
- Service scoping defines the scope of a business process as an aggregation of aspects that include:
  - where the process starts and ends;
  - typical customers of the process;
  - inputs and outputs that customers expect to see;
  - external entities that the process is expected to interface with;
  - different types of events that start an instance of the process.

# Service identification

RosettaNet PIP3A4 “Manage Purchase Order”




***PIP3C “Returns and Finance”***

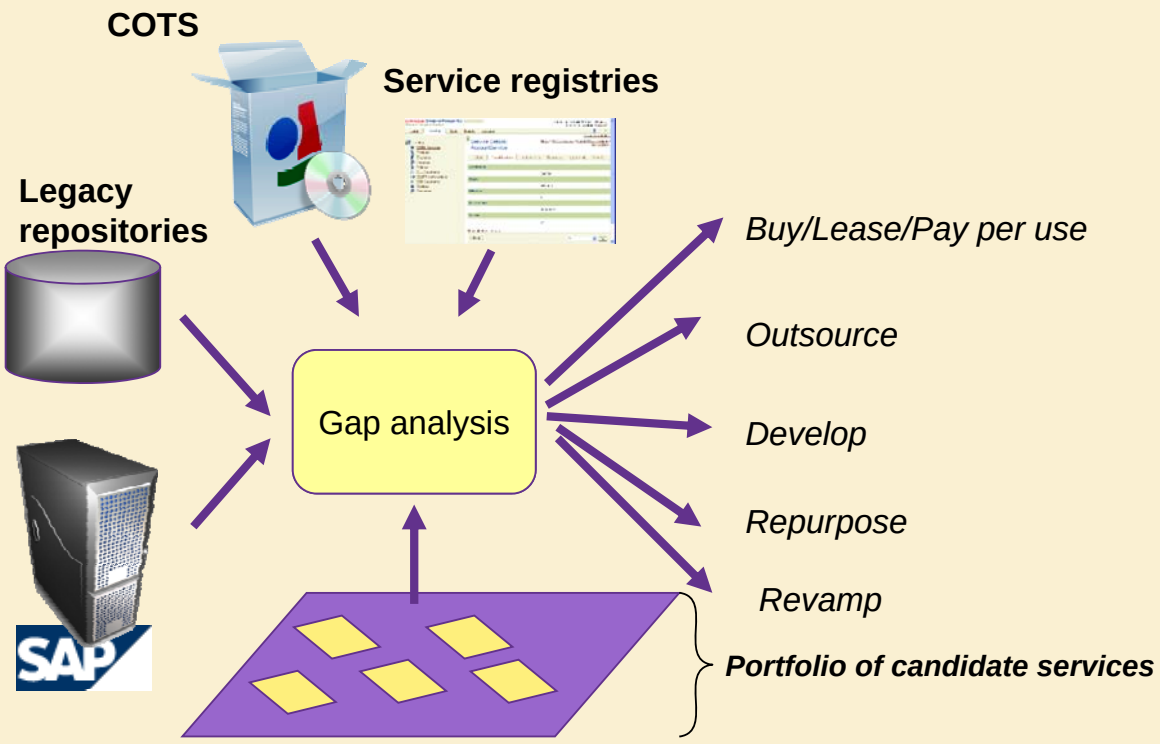
PIP3A, 3C → Order management

Start with comparing the abstract business process portfolio with standard process definitions, e.g., RosettaNet.

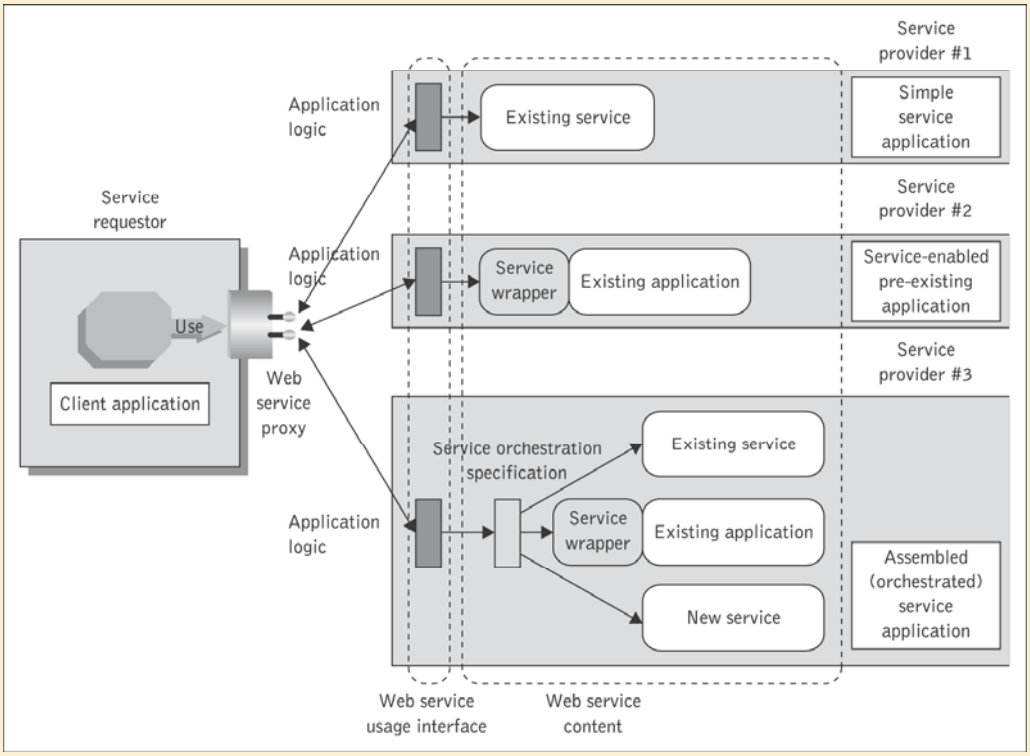




# Business service gap analysis



# Realization architecture and services portfolio








# Business service realization analysis

- Green-field development: describes how new interface for a service will be created.
- Top-down development: a new service can be developed that conforms to an existing service interface.
- Bottom-up development: a new service interface is developed for an existing application.
- Meet-in-the-middle development: this option is used when an already existing service interface – for which an implementation already exists – is partially mapped onto a new service or process definition.

# Service realization alternatives

1. Reusing or repurposing already existing Web services, business processes, or business process logic.
2. Developing new Web services or business processes logic from scratch.
3. Purchasing/leasing/paying per use for services.
4. Outsourcing service design and implementation of Web services or (parts of) business processes.
5. Using wrappers and/or adapters to revamp existing enterprise (COTS) components or existing (ERP/legacy) systems.
    - Revamping software components including database functionality or legacy software results in introducing service-enabling implementations for these systems in the form of adapters or wrappers.





# Topics

- *Web services development*
- *Properties of service development methodology*
- *Qualities of service development methodology*
- *Web services development lifecycle*
- *Service analysis*
- *Service design*
- *Service construction*
- *Service provisioning*
- *Service deployment*

# Service design concerns

- Service design requires developers to define related, well-documented interfaces for all conceptual services identified by the analysis phase, prior to constructing them.
- The design phase encompasses the steps of:
  - singular service specification;
  - business process specification; and
  - policy specification for both singular services and business processes.
- Service design is based on a twin-track design approach that provides two production lines – one along the logical part and one along the physical part of the SOA – and considers both functional and non-functional service characteristics.





# Service design

- Concerns: design for reuse, design for composition, and granularity.
- Specification: structural specification, behavioral specification, service programming style, and policy specification.
- Choice of “right” standards, e.g., orchestration versus choreography.

# Specifying service types

```
<wsdl:types>
      <xsd:complexType name = "PIP3A4PurchaseOrderRequest">
            <xsd:sequence>
                  <xsd:element ref = "PurchaseOrder"/>
                  <xsd:element ref = "fromRole"/>
                  <xsd:element ref = "toRole"/>
                  <xsd:element ref = "thisDocumentGenerationDateTime"/>
                  <xsd:element ref = "thisDocumentIdentifier"/>
                  <xsd:element ref = "GlobalDocumentFunctionCode"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name = "PurchaseOrder">
            <xsd:sequence>
                  <xsd:element ref = "deliverTo" minOccurs = "0"/>
                  <xsd:element ref = "comment" minOccurs = "0"/>
                  <xsd:element ref = "packListRequirements" minOccurs = "0"/>
                  <xsd:element ref = "ProductLineItem" maxOccurs = "unbounded"/>
                  <xsd:element ref = "GlobalShipmentTermsCode"/>
                  <xsd:element ref = "RevisionNumber"/>
                  <xsd:element ref = "prePaymentCheckNumber" minOccurs = "0"/>
                  <xsd:element ref = "QuoteIdentifier" minOccurs = "0"/>
                  <xsd:element ref = "WireTransferIdentifier" minOccurs = "0"/>
                  <xsd:element ref = "AccountDescription" minOccurs = "0"/>
                  <xsd:element ref = "generalServicesAdministrationNumber"
                                    minOccurs = "0"/>
                  <xsd:element ref = "secondaryBuyerPurchaseOrderIdentifier"
                                    minOccurs = "0"/>
                  <xsd:element ref = "GlobalFinanceTermsCode"/>
                  <xsd:element ref = "PartnerDescription" maxOccurs = "unbounded"/>
                  <xsd:element ref = "secondaryBuyer" minOccurs = "0"/>
                  <xsd:element ref = "GlobalPurchaseOrderTypeCode"/>
            </xsd:sequence>
      </xsd:complexType>
</wsdl:types>
  … … … …
<message name="PurchaseOrderRequest">
      <part name="PO-body" type="tns:PIP3A4PurchaseOrderRequest"/>
</message>
```

Type definitions for POs and PO requests in RosettaNet.





# Specifying service interfaces

```
   … … … …
<portType name="CanReceive3A42_PortType">
      <!-- name of operation is same as name of message -->
      <operation name="PurchaseOrderRequest">
            <output message="tns:PurchaseOrderRequest"/>
      </operation>
      <operation name="ReceiptAcknowledgement">
            <output message="tns:ReceiptAcknowledgment"/>
      </operation>
</portType>

<portType name="CanSend3A42_PortType">
      <!-- name of operation is same as name of message -->
      <operation name="PurchaseOrderConfirmation">
            <input message="tns:PurchaseOrderConfirmation"/>
      </operation>
      <operation name="ReceiptAcknowledgment">
            <input message="tns:ReceiptAcknowledgment"/>
      </operation>
      <operation name="Exception">
            <input message="tns:Exception"/>
      </operation>
</portType>
    … … … …
```

Interface definitions for PO requests in RosettaNet.

# Specifying business processes

- Describe the process structure:
  - Identify and group activities that implement a business process;
  - Describe activity dependencies, conditions, or synchronization;
  - Describe implementation of the business process.
- Describe roles.
- Capture non-functional process concerns, e.g., security and transactions.



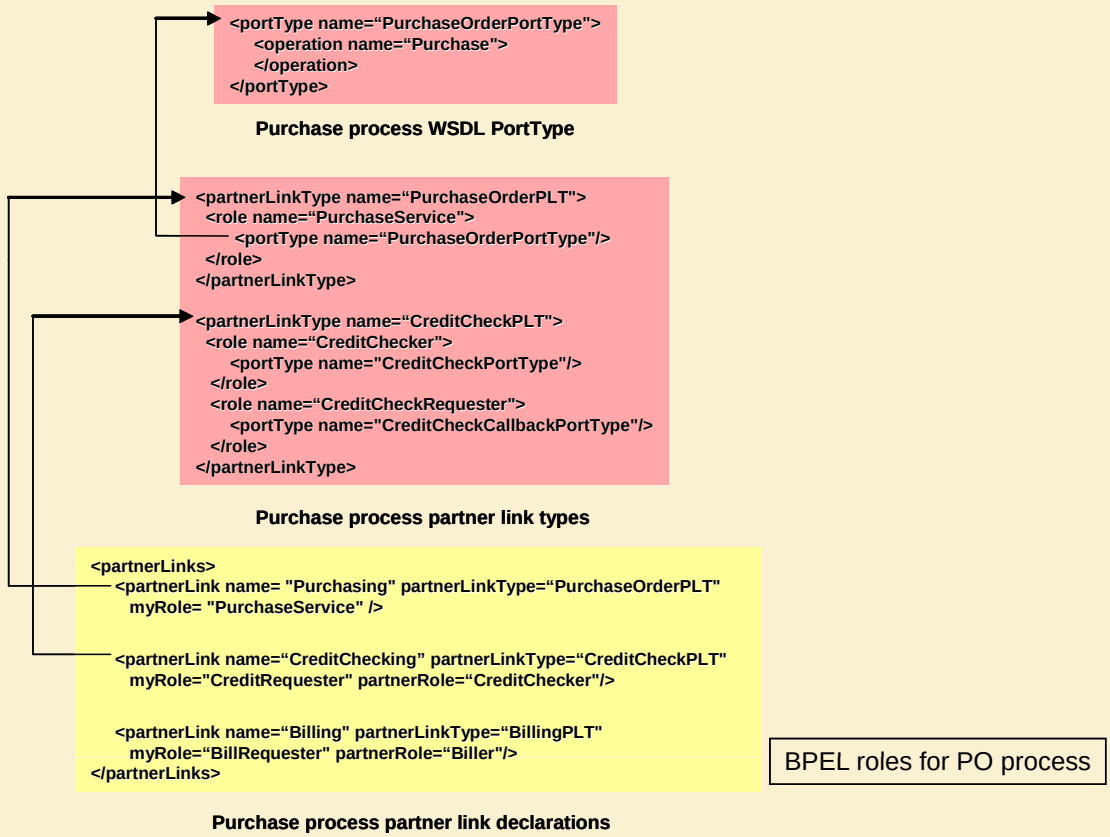


# Process flow

```
<sequence>
    <receive   partner="Manufacturer" portType="lns:PurchaseOrderPortType"
                    operation="Purchase"   variable="PO"
                    createInstance="yes" >
   </receive>
 <flow>
     <links>
          <link name="inventory-check"/>
          <link name="credit-check"/>
     </links>
    <!– Check inventory –->
    <invoke partner="inventoryChecker"
                portType="lns:InventoryPortType"
                operation="checkInventory"
                … … …
            <source linkName="inventory-check"/>
        …
    <invoke/>
    <!– Check  credit –->
    <invoke partner="creditChecker"
                portType="lns:CreditCheckPortType"
                operation="checkCredit"
                … … …
            <source linkName="credit-check"/>
        ….
    <invoke/>
    <!– Issue bill once inventory and credit checks are succesful -->
    <invoke partner="BillingService"
                portType="lns:BillingPortType"  operation="billClient"
                inputVariable="billRequest"  outputVariable="Invoice" >
                joinCondition="getLinkStatus("inventory-check") AND
                                    getLinkStatus("credit-check")" />
            <target linkName="inventory-check"/>
            <target linkName="credit-check"/>
        ….
    <invoke/>
 </flow>
   ...
 <reply partnerLink="Purchasing" portType="lns:purchaseOrderPT"
                          operation="Purchase" variable="Invoice"/>
</sequence>
```

BPEL process flow for PO process

# Defining roles




BPEL roles for PO process





# Specifying service policies

Service policies could specify three sets of constraints:

- *Business constraints:* such as operating ranges, regulatory and legal constraints, or standards established in specific vertical industries.
- *Technology constraints:* based on choices, decisions, and commitments to specific technologies in current and continued use in the enterprise infrastructure.
  - e.g., choice of specific application packages, legacy applications, commitments to industry specific standards, etc.
- *Run-time quality constraints:* services aspects that are directly related to system dynamics e.g., performance, scalability, transactional integrity, security, and fault tolerance.
  - In SOAs run-time qualities are captured by SLAs.

# Services integration model

- A services integration model facilitates the design of a service integration strategy to solve integration and interoperability problems mainly between interacting enterprises. This strategy considers:
  - service design models, policies, SOA governance options, and organizational and industry best practices and conventions.
- The SOA integration model is based on service integration principles.
  - *Service relationship principle:* regards services relationships founded in business processes in terms of service producers and consumers during service design.
  - *Service transportation principle:* regards message interceptors or brokers interposed between service consumers and providers.
  - *Service delivery principle:* describes the mechanisms which service intermediaries, consumers, and producers use to deliver messages to endpoints.
  - *Process flow principle:* describes how service conversations and behavioral activities in which service providers and consumers are engaged in are influenced by business, technical, and environmental requirements.

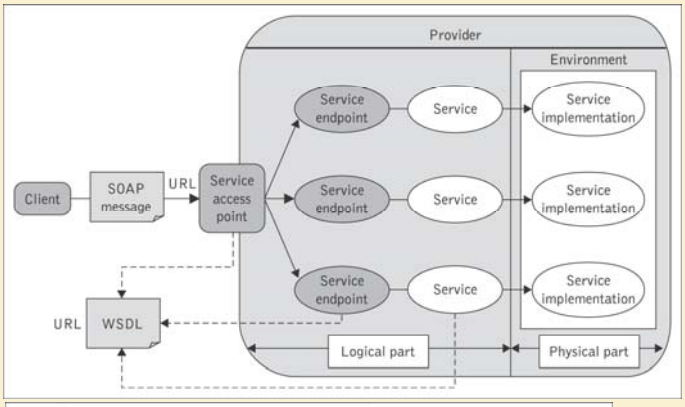




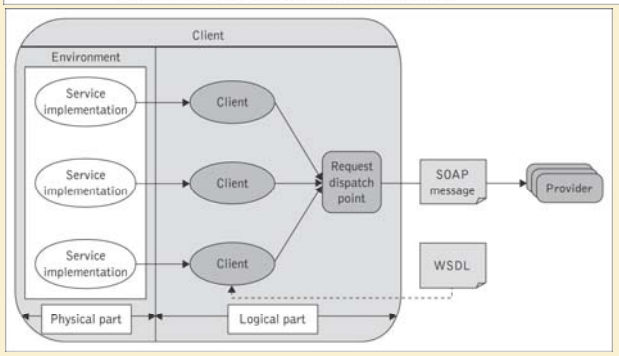
# Topics

- *Web services development*
- *Properties of service development methodology*
- *Qualities of service development methodology*
- *Web services development lifecycle*
- *Service analysis*
- *Service design*
- *Service construction*
- *Service provisioning*
- *Service deployment*

# Constructing a service




The provider perspective




The client perspective





# Topics

- *Web services development*
- *Properties of service development methodology*
- *Qualities of service development methodology*
- *Web services development lifecycle*
- *Service analysis*
- *Service design*
- *Service construction*
- *Service provisioning*
- *Service deployment*

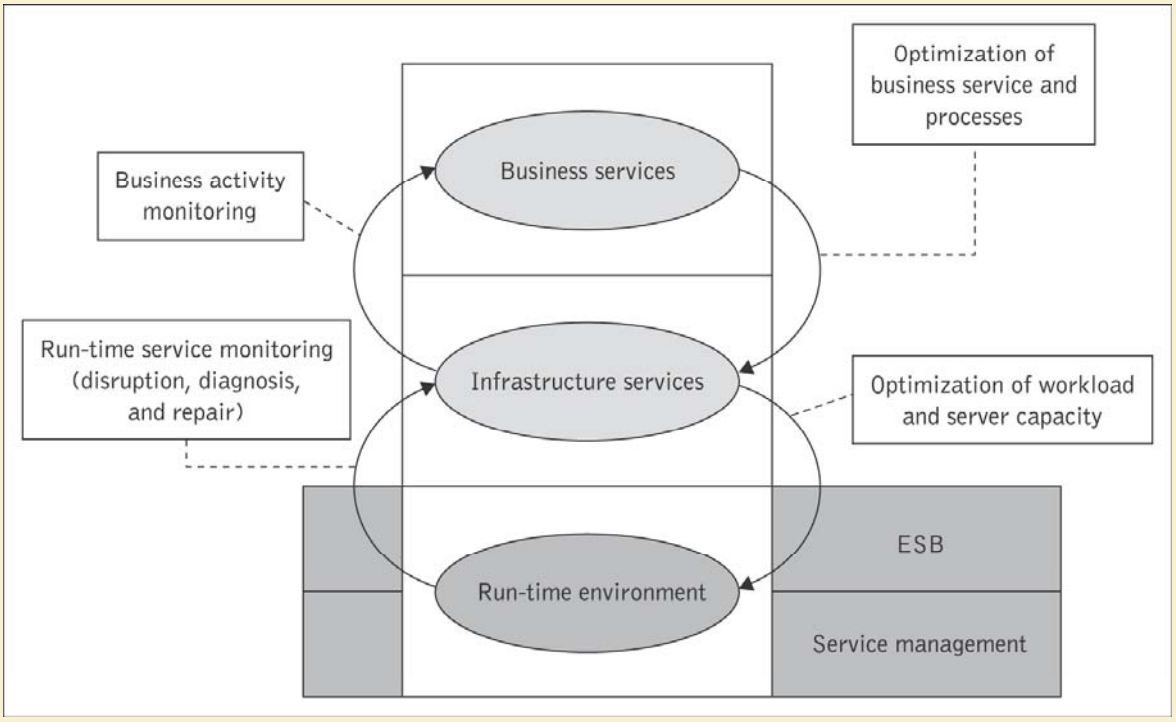
# SOA governance

- Services that flow between enterprises have defined owners with established ownership and governance responsibilities, including gathering requirements, design, development, deployment, and operations management for *any mission-critical or revenue-generating service*.
- SOA governance refers to the organization, process, policies, and metrics that are required to manage an SOA successfully.
- SOA governance is a formalization of the structured relationships, procedures, and policies that ensure the IT architecture in an organization supports and is aligned to business functions, with a specific focus on the life cycle of services. SOA governance is primarily designed to:
  - enable enterprises to maximize business benefits of SOA such as increased process flexibility, improved responsiveness, and reduced IT maintenance costs.
- Two different governance models are possible: central governance versus federated governance.





# The role of the ESB and SOA governance



# Topics

- *Web services development*
- *Properties of service development methodology*
- *Qualities of service development methodology*
- *Web services development lifecycle*
- *Service analysis*
- *Service design*
- *Service construction*
- *Service provisioning*
- *Service deployment*





# Service deployment phase

- Green-field services deployment steps:
  - Publish the service interface
  - Deploy the Web service
  - Create the service implementation definition
  - Publish the service implementation definition
- Top-down services deployment steps:
  - Deploy the Web service
  - Create the service implementation definition
  - Publish the service implementation definition
- Bottom-up services deployment steps:
  - Deploy the Web service
  - Create the service implementation definition
  - Publish the service interface definition
  - Publish the service implementation definition