Please comply with epidemic prevention and control policies

- Choose the seat that you will use during the whole semester and register your information in the system.
- Make sure to take the same seat during every lecture.

Reminder
Hot topic study – task 3 – individual/group

- Create a home page in your groups wiki area devoted to your selected topic – give an overview of your topic
- Extended deadline: 15th May

Reminder
Case study assignment

- Read the case study → (document in the Blackboard)
- Answer questions 3 and 4– submit to the Blackboard by May 8th
  - 3.Follow the service-oriented analysis and service-oriented design stages of the SOA delivery lifecycle to design an example services of your choice for Bama Tea.
  - 4.Suggest suitable development technologies.

COMP3017
Service Computing

Review

Final Examination format

- Multiple choice questions – 10 questions, 3 points each
- Short description questions – 5 questions, 6 points each
- Case questions – 4 questions, 40 points in total

Multiple choice questions

- One question - Multiple answers options
- But only ONE answer is correct

- Following are the topics for your review
- In addition
  - Have a look at our lecture slides where more topics were discussed (I indicated the important topics in red)
  - Have a look at all the questions that were sent in Rain Classroom

Module One: Introduction to Service Computing and XML-RPC

Do you know what these acronyms mean?

BPEL

SOAP

UDDI

JEE

WSDL

XML-RPC

SOA

Guided questions for Module 1

- What is a service?
- What is a web-service?
- What is a web-service protocol stack?
- What is XML messaging?
- What is service description?
- What is service discovery?
- What is XML-RPC?
- What is SOAP?
- What is WSDL?
- What is UDDI?

Guided questions for Module 1

- What is the relationship between XML, XML-RPC, and SOAP?
- What is the relationship between SOAP, WSDL, and UDDI?
- What is the relationship between WSDL and service description?
- What is the relationship between UDDI and service discovery?
- What are the three main actors in service architecture?
- What are the steps in a typical development plan for a service requestor?
- How do we develop web services from the service provider perspective?

Guided questions for Module 1

- What data types are defined in XML-RPC specification and how are they represented?
- What elements are in XML-RPC request?
- What elements are in XML-RPC response?
- How developer uses XML-RPC?

Service

- Services represent a type of relationships-based interactions (activities) between at least one service provider and one service consumer to achieve a certain business goal or solution objective.

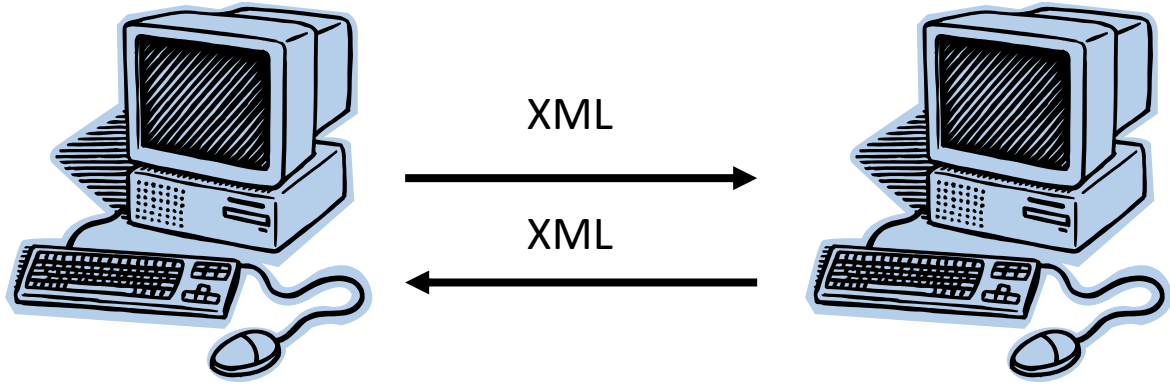
What is Service Computing?

- Services Computing is a cross-discipline that covers the science and technology of bridging the gap between business services and IT services.
- Supports integrating the business as linked, repeatable business tasks, or services.

What is a Web Service?

- A Web Service is any service that:
  - Is available over the Internet or private (intranet) networks
  - Uses a standardized XML messaging system
  - Is not tied to any one operating system or programming language

A Basic Web Service



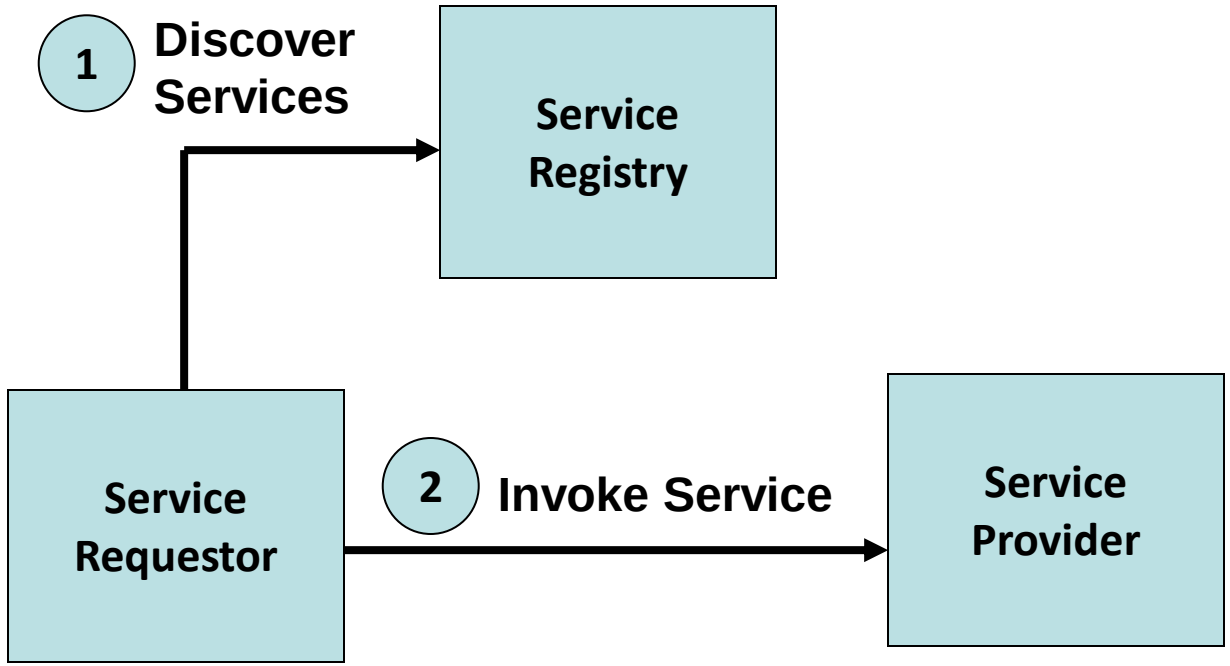
Computer A:
Language: Perl
Operating System: Windows 2000

Computer B:
Language: Java
Operating System: Linux

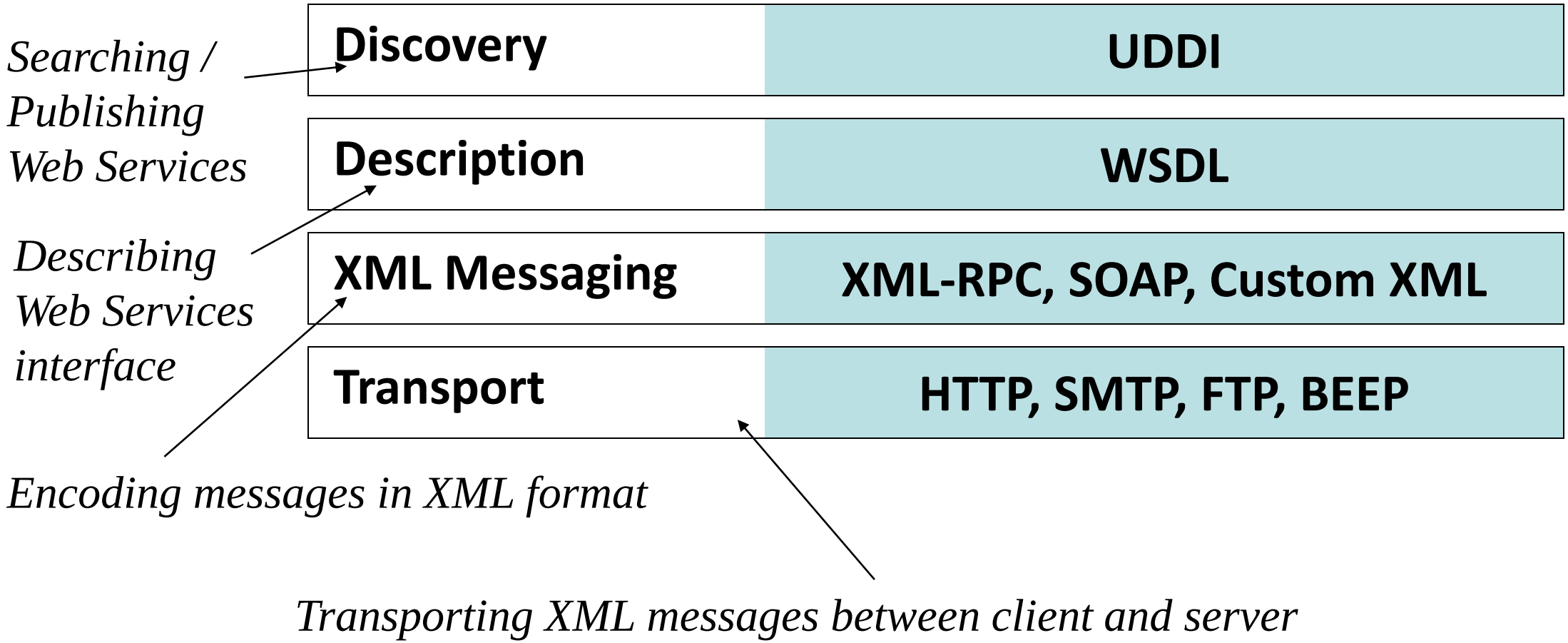
Web Services Defined

- Although not required, a web service may also have two additional (and desirable) properties:
  - a web service should be *self describing*.
  - a web service should be *discoverable*.

Web Service Roles



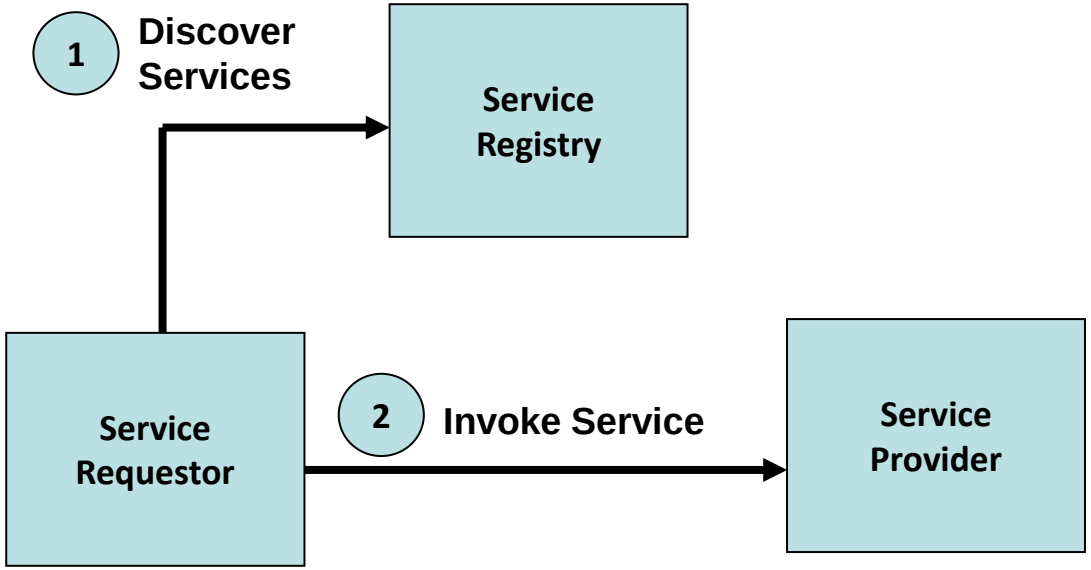
Web Service Protocol Stack



XML Messaging

Discovery	UDDI
Description	WSDL
XML Messaging	XML-RPC, SOAP, Custom XML
Transport	HTTP, SMTP, FTP, BEEP

Encoding messages in XML format



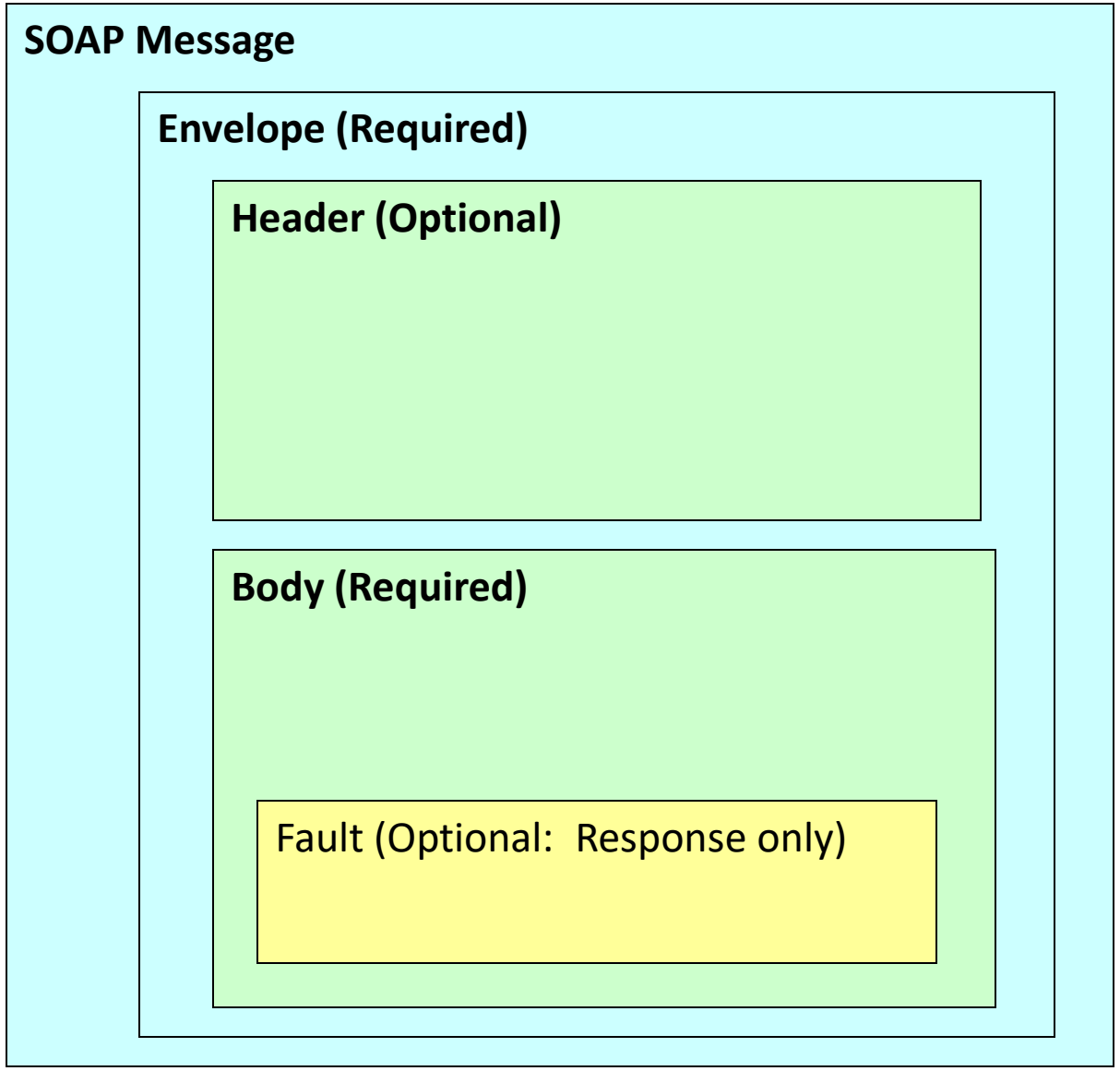
Option 1: XML-RPC

- XML-RPC: protocol that uses XML messages to perform Remote Procedure Calls (RPC.)
- Platform independent; diverse applications can talk to each other.
- XML-RPC is the easiest way to get started with web services.
  - Simpler than SOAP
  - Simpler data structures for transmitting data.

Option 2: SOAP

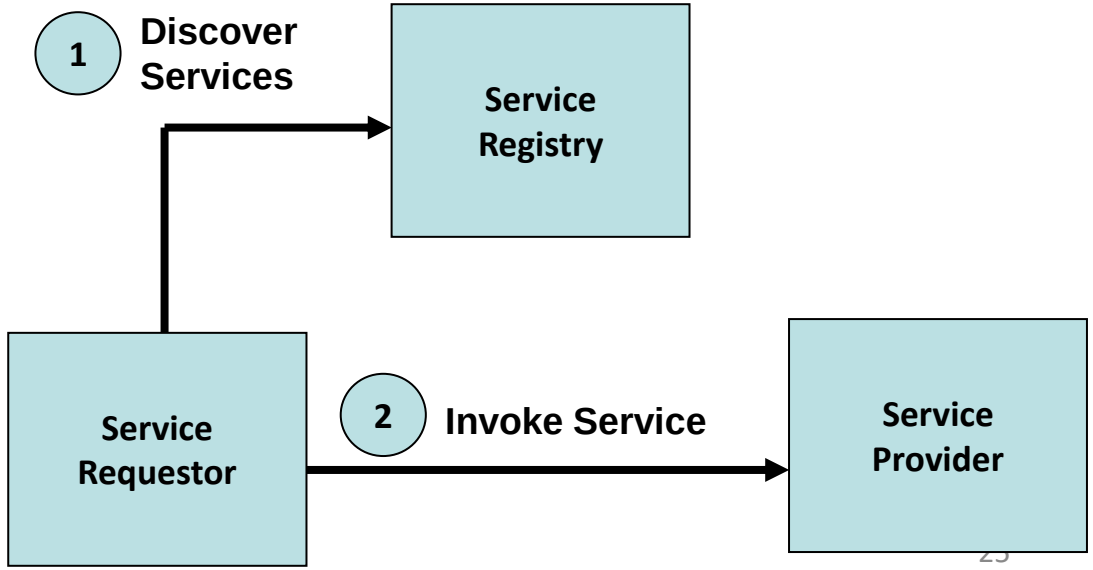
- SOAP: used to stand for “Simple Object Access Protocol”
- XML-Based protocol for exchanging information between computers.
- Currently a formal recommendation of the World Wide Web Consortium (W3C.)

SOAP Message Format



Discovery	UDDI
Description	WSDL
XML Messaging	XML-RPC, SOAP, Custom XML
Transport	HTTP, SMTP, FTP, BEEP

Describing Web Services interface



WSDL

- WSDL: Web Service Description Language.
- WSDL is an XML grammar for specifying an interface for a web service.
- Specifies
  - location of web service
  - methods that are available by the web service
  - data type information for all XML messages
- WSDL is commonly used to describe SOAP services.

WSDL In a Nutshell

<definitions>: Root WSDL Element

- **<types>: What data types will be transmitted?**
- **<message>: What messages will be transmitted?**
- **<portType>: What operations (functions) will be supported?**
- **<binding>: What SOAP specific details are there?**
- **<service>: Where is the service located?**

WSDL Excerpt: Weather Service

```
<message name="getWeatherRequest">
  <part name="zipcode" type="xsd:string"/>
</message>
<message name="getWeatherResponse">
  <part name="temperature" type="xsd:int"/>
</message>

<portType name="Weather_PortType">
  <operation name="getWeather">
    <input message="tns:getWeatherRequest"/>
    <output message="tns:getWeatherResponse"/>
  </operation>
</portType>
```

Can you identify abstract and concrete parts of an example WSDL document?

Do you know what operations are available in this web service?

WSDL Excerpt: Weather Service

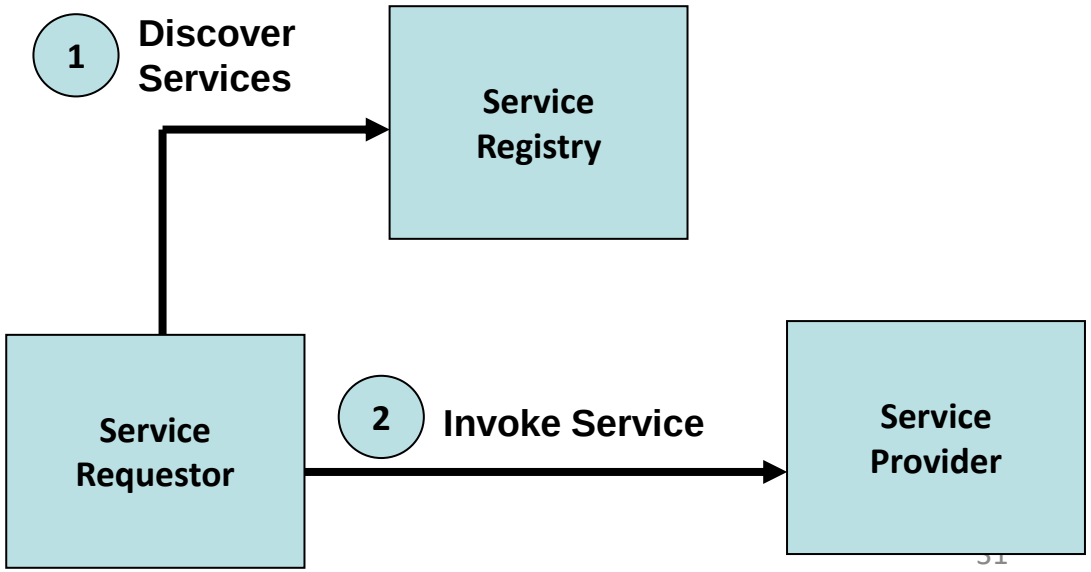
```
…
 <service name="Weather_Service">
     <documentation>WSDL File for
    Weather Service</documentation>
     <port binding="tns:Weather_Binding"
              name="Weather_Port">
              <soap:address
          location="http://ecerami.com/soap/servlet/rpcrouter"/>
     </port>
   </service>
</definitions>
```

What is the use of WSDL?

- Given a WSDL file, a developer can immediately figure out how to connect to the web service.
- Eases overall integration process.
- Better yet, with WSDL tools, you can *automate* the integration…

Searching / Publishing Web Services

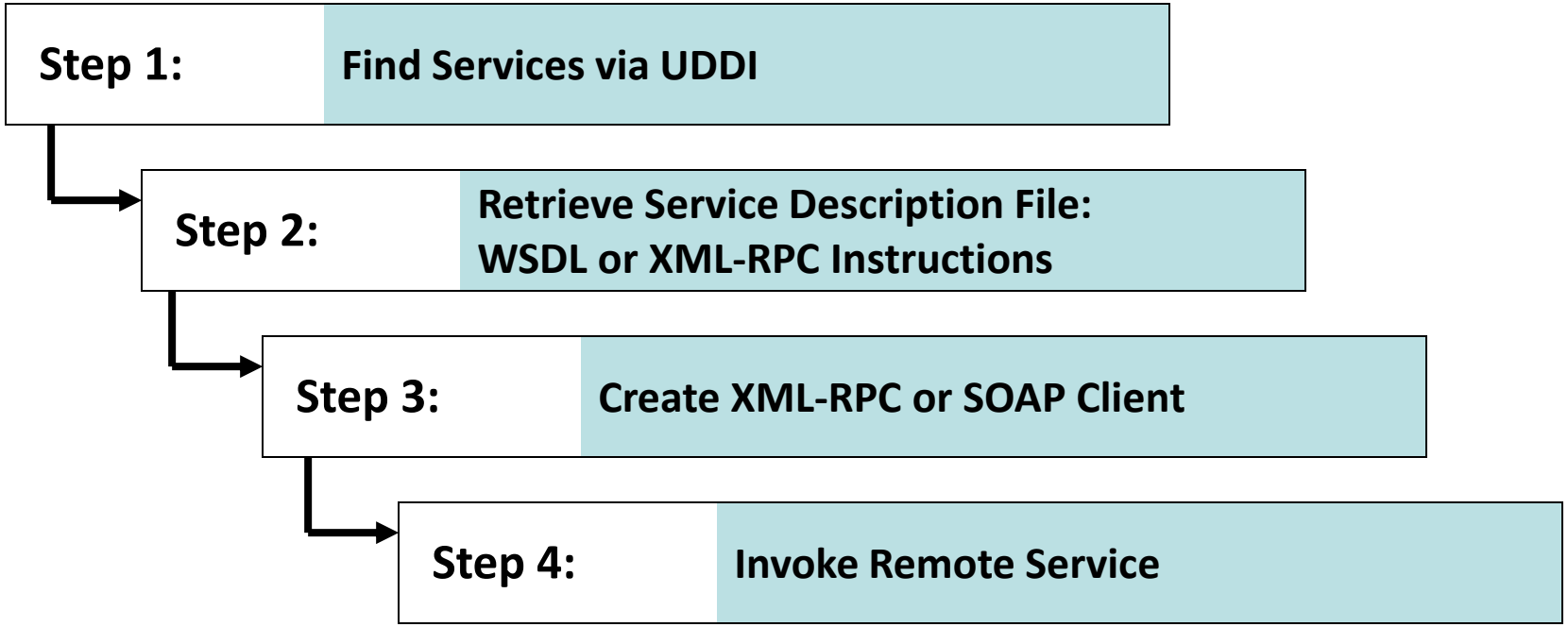
Discovery	UDDI
Description	WSDL
XML Messaging	XML-RPC, SOAP, Custom XML
Transport	HTTP, SMTP, FTP, BEEP



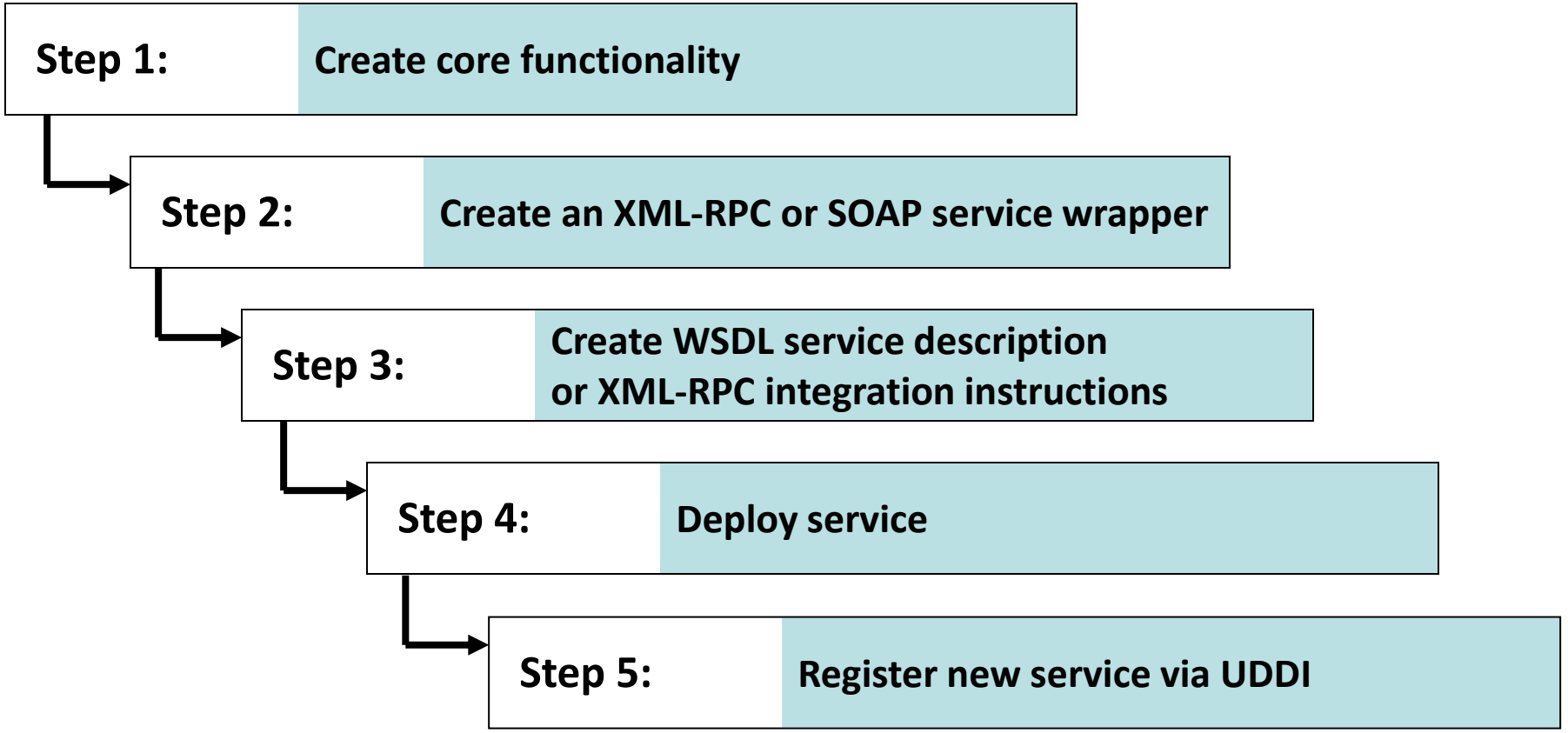
UDDI

- UDDI: Universal Description, Discovery and Integration.
- Currently represents the *discovery* layer in the protocol stack.
- Originally created by Microsoft, IBM and Ariba.
- Technical specification for publishing and finding businesses and web services.

Using the Protocols Together – service request perspective



Using the Protocols Together – service provider perspective



Module Two: Service Message Exchange SOAP and Service Description WSDL

Guided questions for Module 2

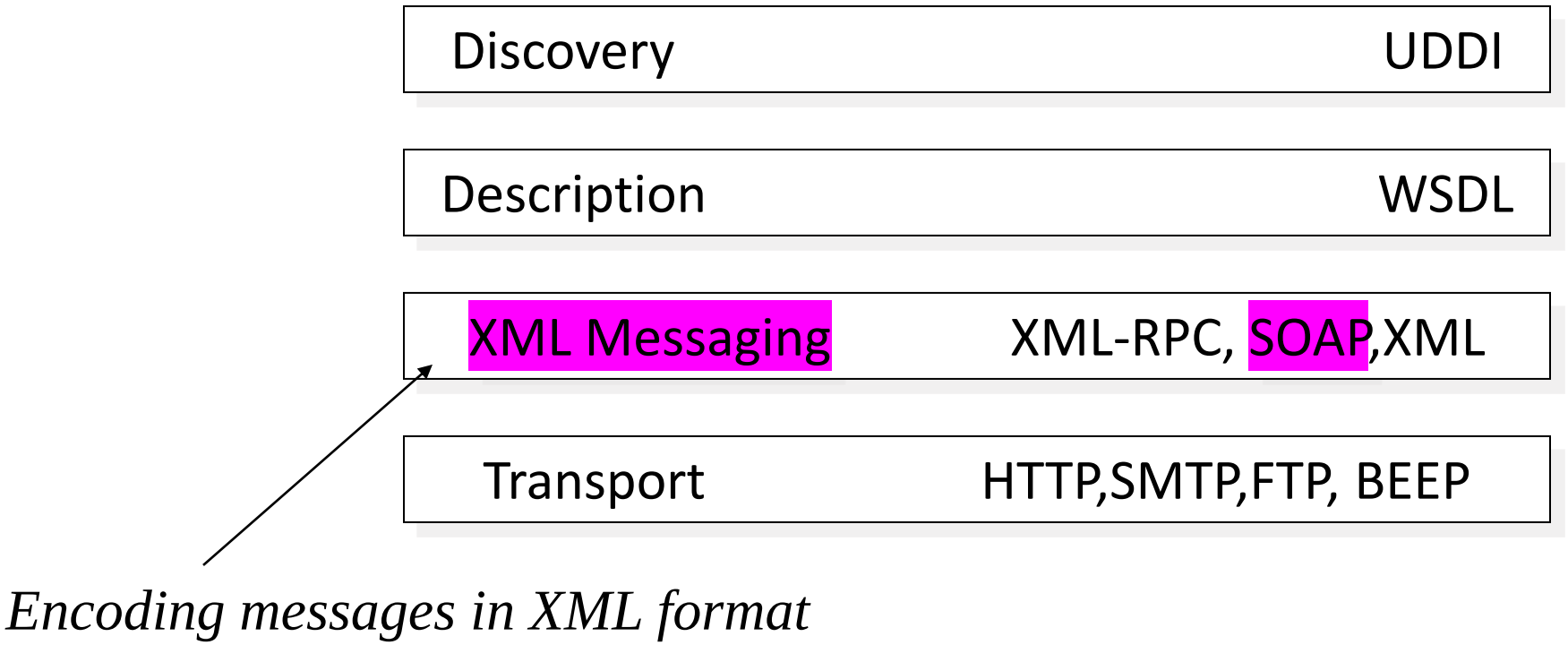
- What is the relationship between XML, XML-RPC, and SOAP?
- What is advantage of SOAP over CORBA, DCOM, and Java RMI?
- What platform and language do we need to use with SOAP? (tricky question)
- What are the major parts in SOAP specification?

Guided questions for Module 2

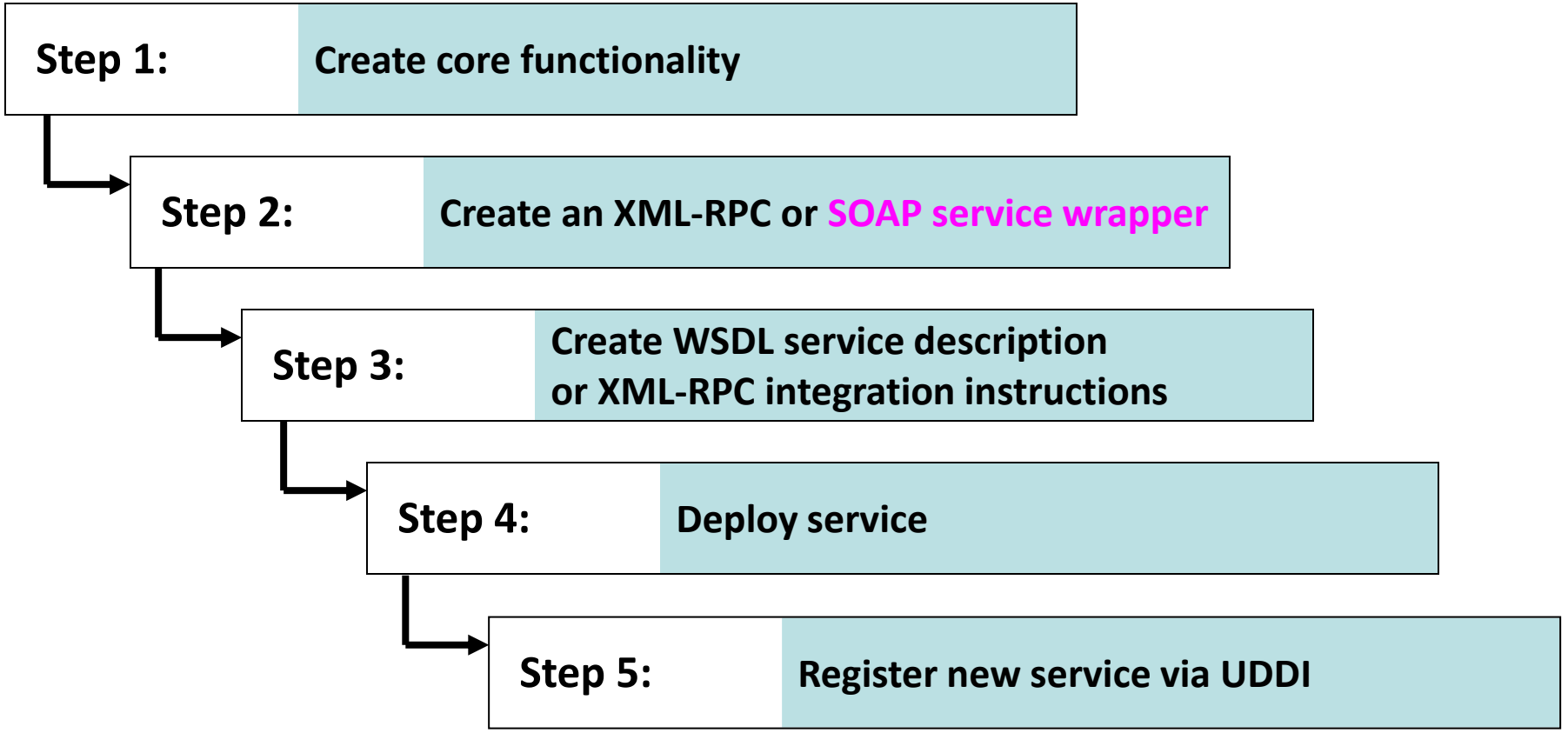
- What is WSDL?
- What is the relationship between WSDL and service description?
- What data does WSDL describe?
- What is WSDL used for?
- What platform and language do we need to use with WSDL? (tricky question)
- What are the major elements of WSDL?
- What is the relationship between SOAP, WSDL, and UDDI?

Service Message Exchange SOAP

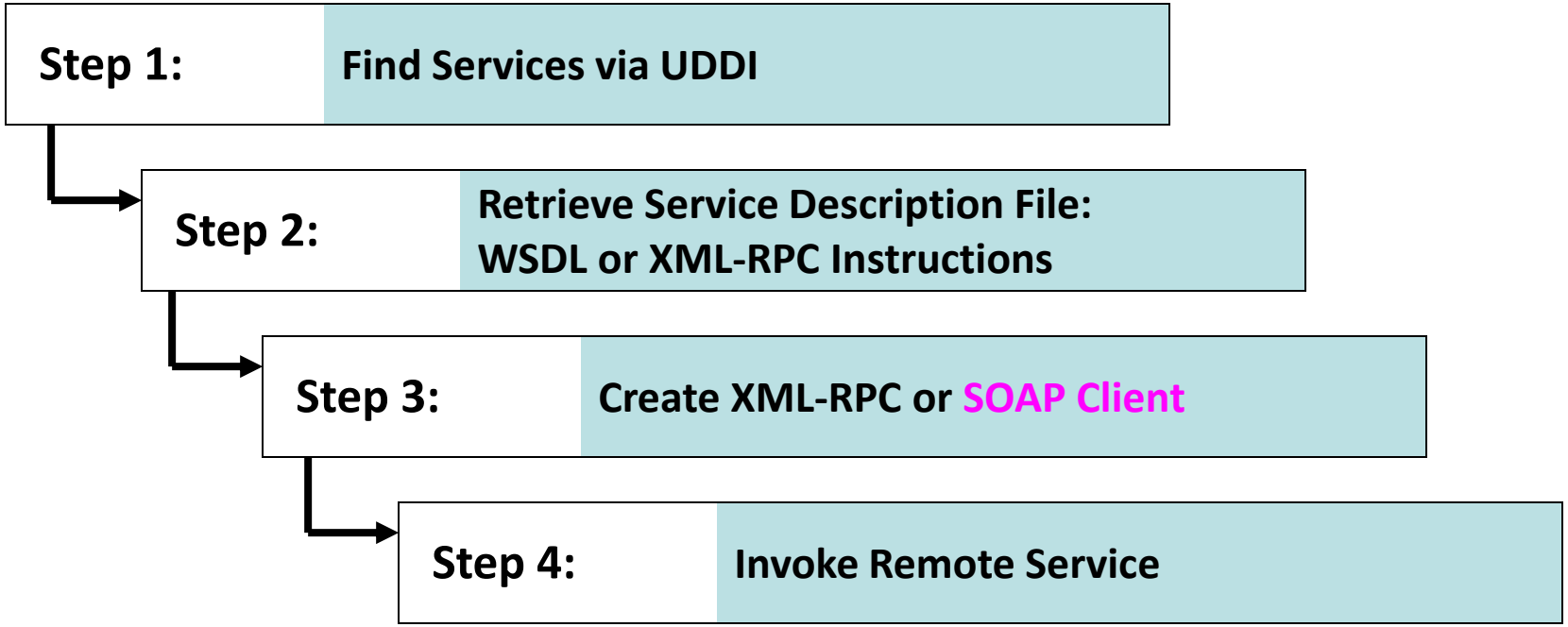
Web Service Protocol Stack



Using the Protocols Together – service provider perspective



Using the Protocols Together – service request perspective



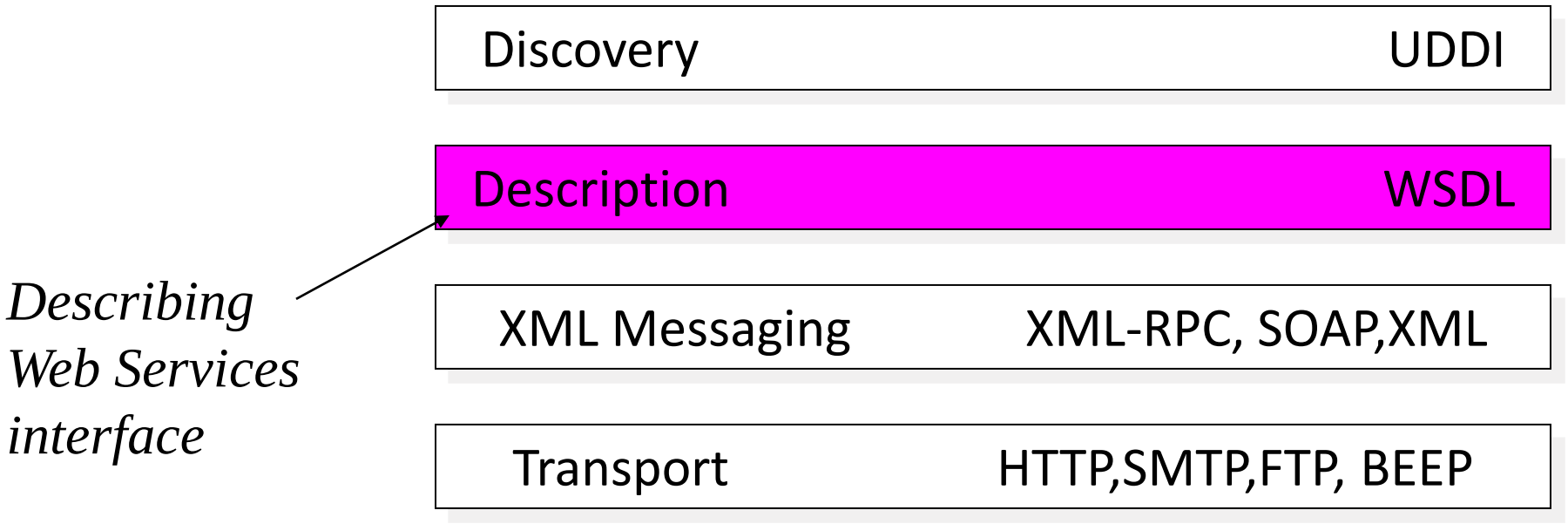
A client program reads a WSDL document to understand what a Web service can do; then it uses SOAP to actually invoke the functions listed in the WSDL document.

SOAP

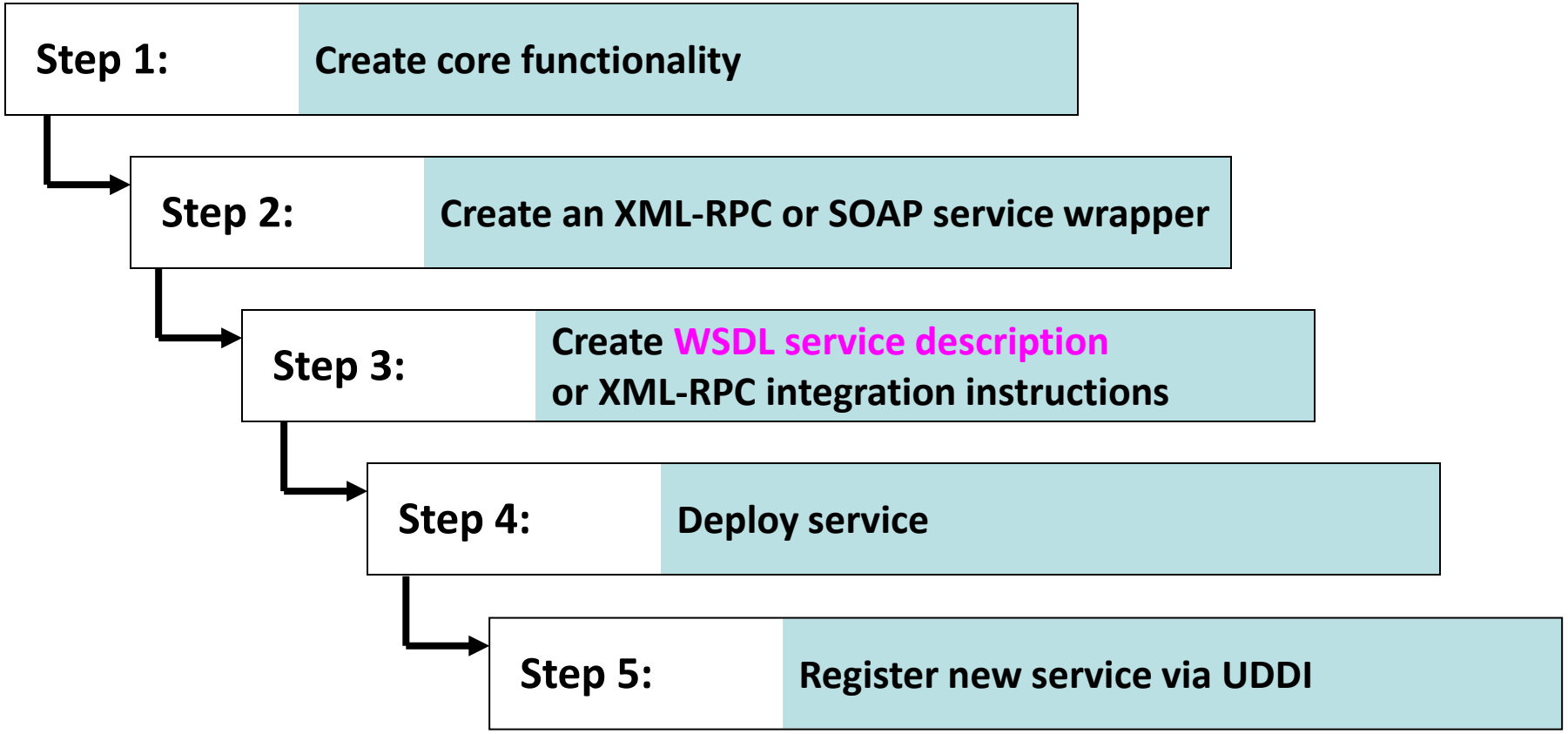
- SOAP is an XML-based protocol for exchanging information between computers.

Service Description
WSDL

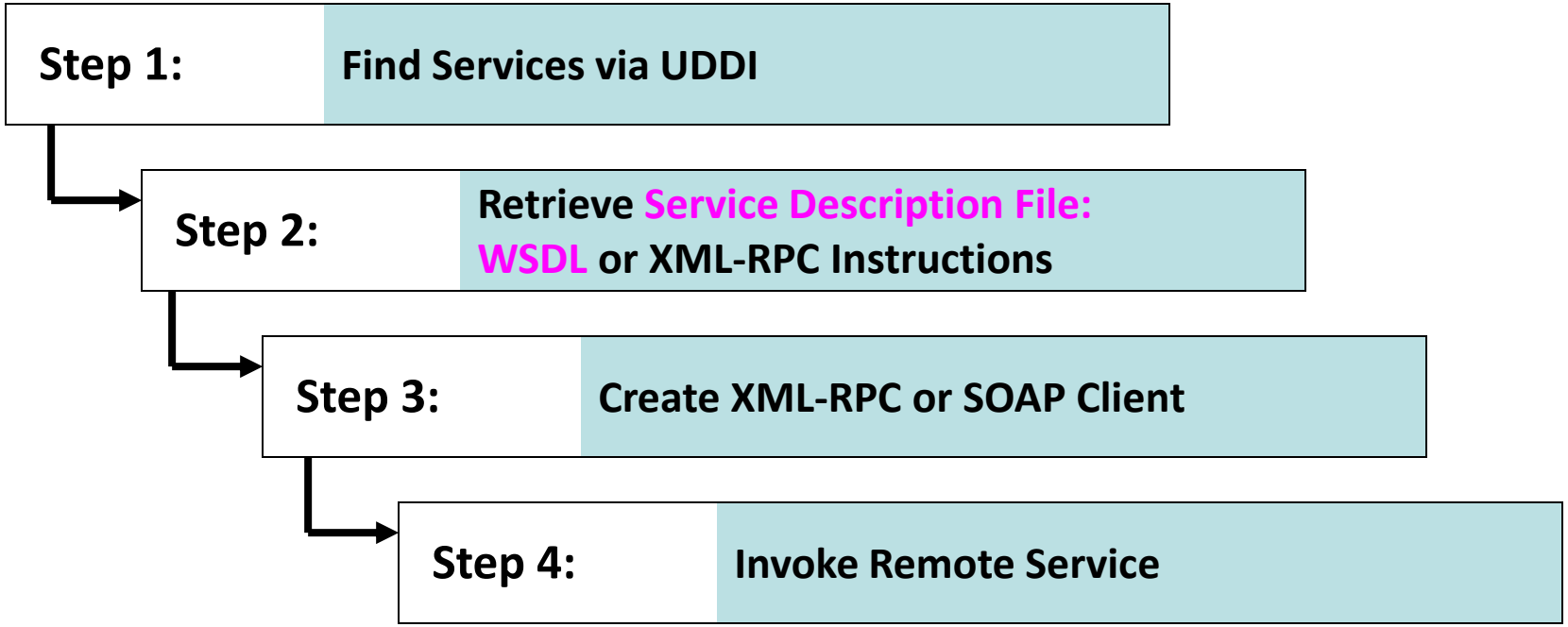
Web Service Protocol Stack



Using the Protocols Together – service provider perspective



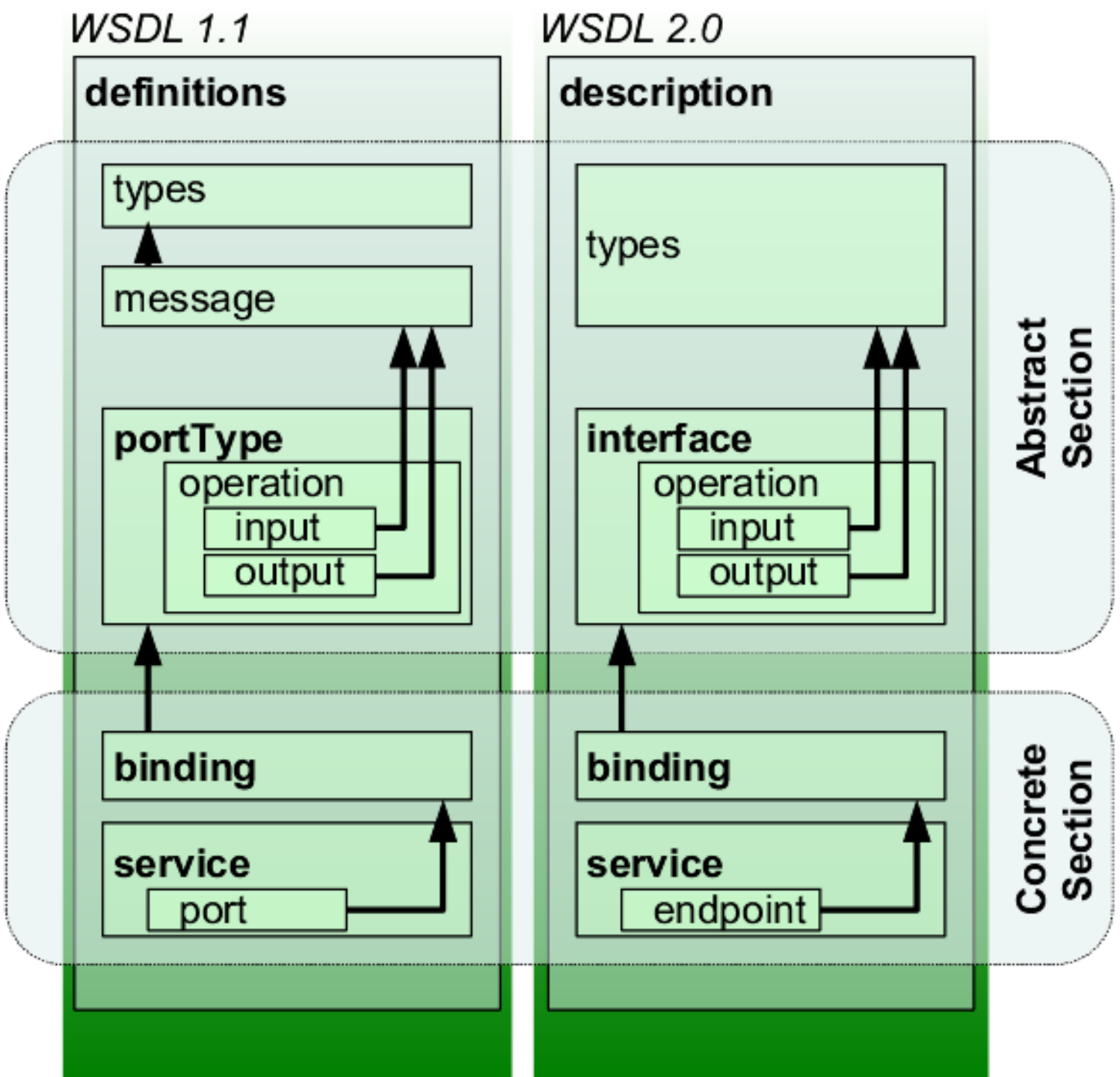
Using the Protocols Together – service request perspective



A client program reads a WSDL document to understand what a Web service can do; then it uses SOAP to actually invoke the functions listed in the WSDL document.

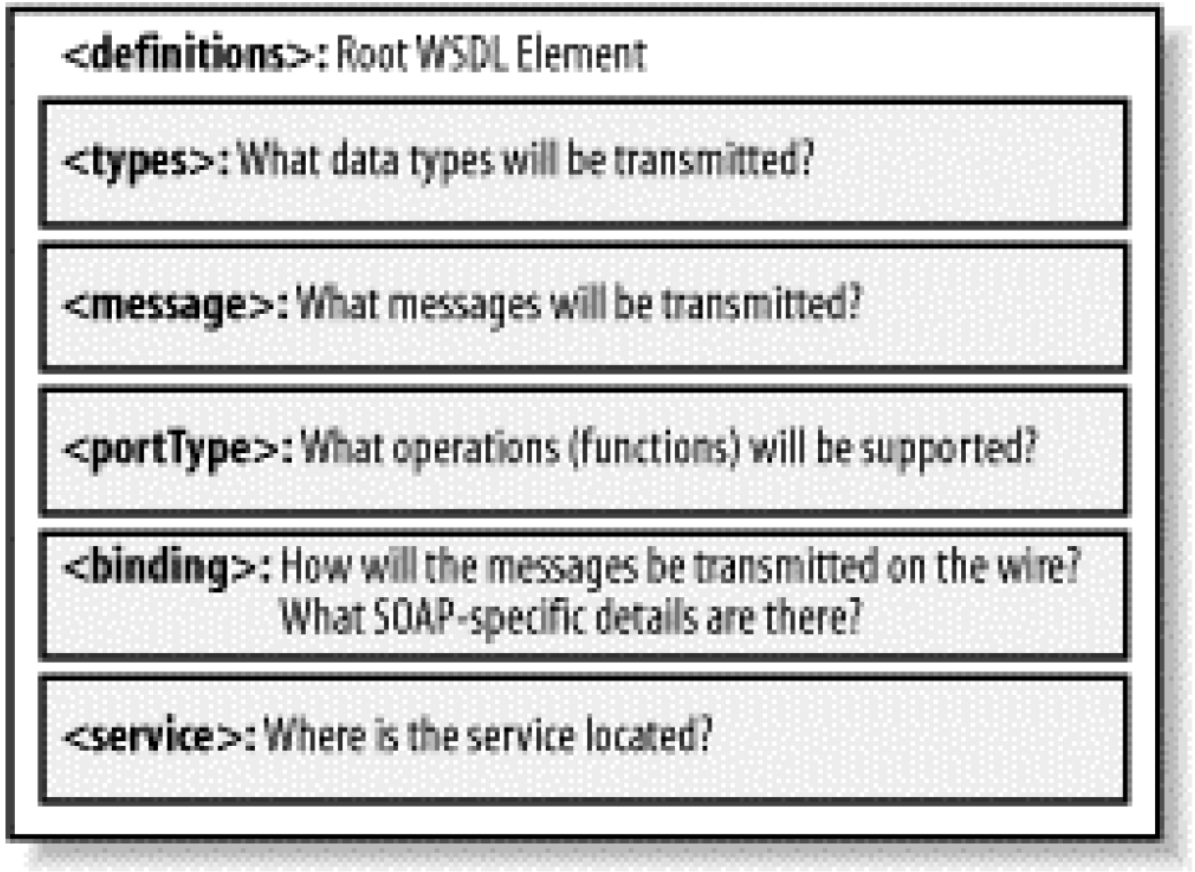
WSDL

- An XML-based interface description language
- Used for describing the functionality offered by a web service
- WSDL describes services as collections of network endpoints or ports



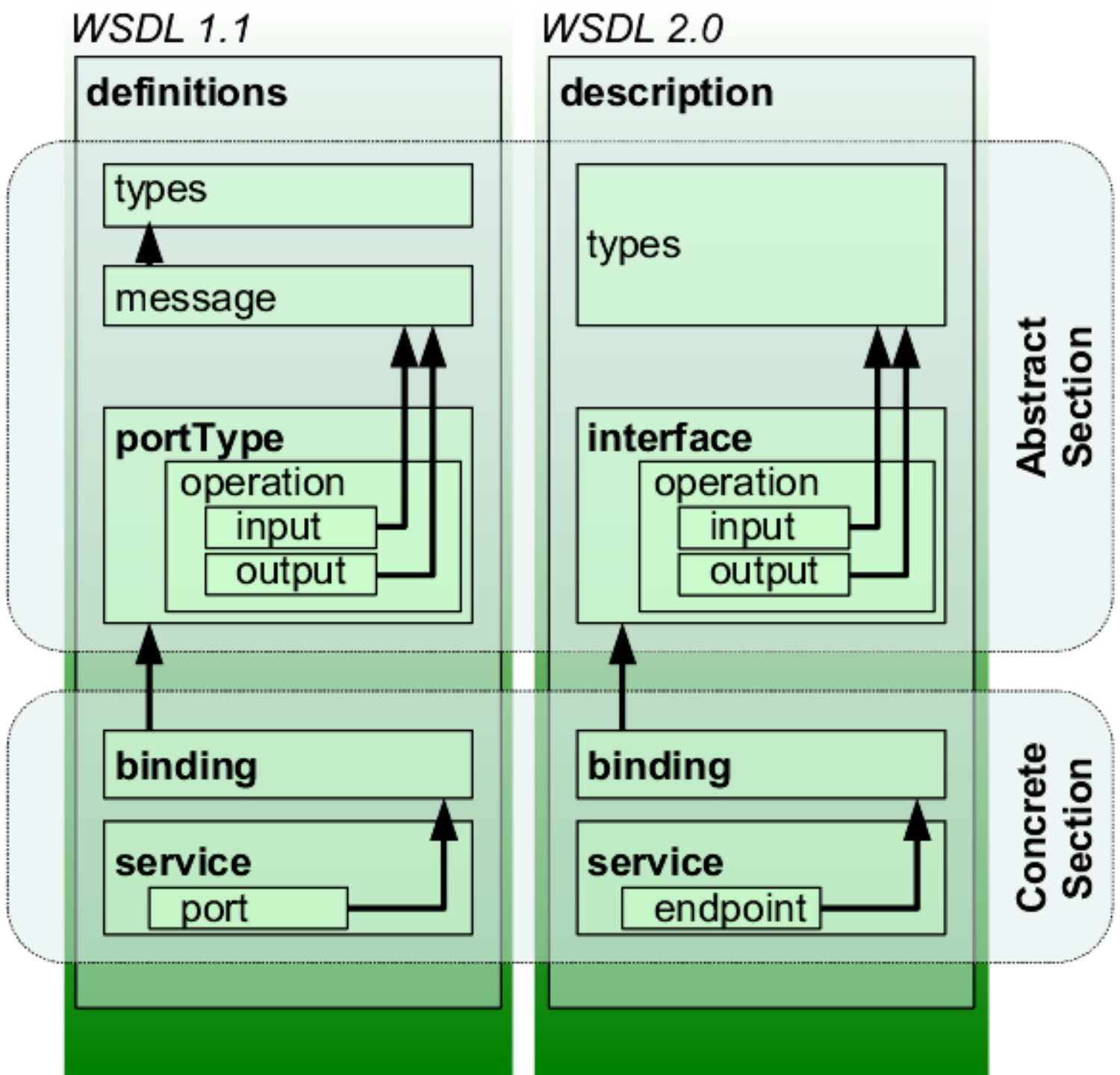
WSDL Specification major Elements

- definitions
- types
- message
- portType
- binding
- service



WSDL

- Abstract and concrete sections



What is WSDL?

- Web service is described as
  - A set of communication endpoints (ports)
- Endpoint is made of two parts
  - Abstract definitions of operations and messages
  - Concrete binding to networking protocol (and corresponding endpoint address) and message encoding
- These portions are often defined in two or more files
  - Concrete file imports the abstract one
- Why this separation?
  - Enhance reusability

Authoring Style Recommendation

- To enhance **Reusability** and **maintainability**
- Maintain WSDL document in 3 separate parts
  - Data type definitions
  - Abstract definitions
  - Specific service bindings
- Use "*import*" element to import necessary part of WSDL document

- In addition, when we studied Module 6, we asked a question: “How web services support SOA design principles?” – part of this question can be answered by pointing out the separate parts of WSDL document

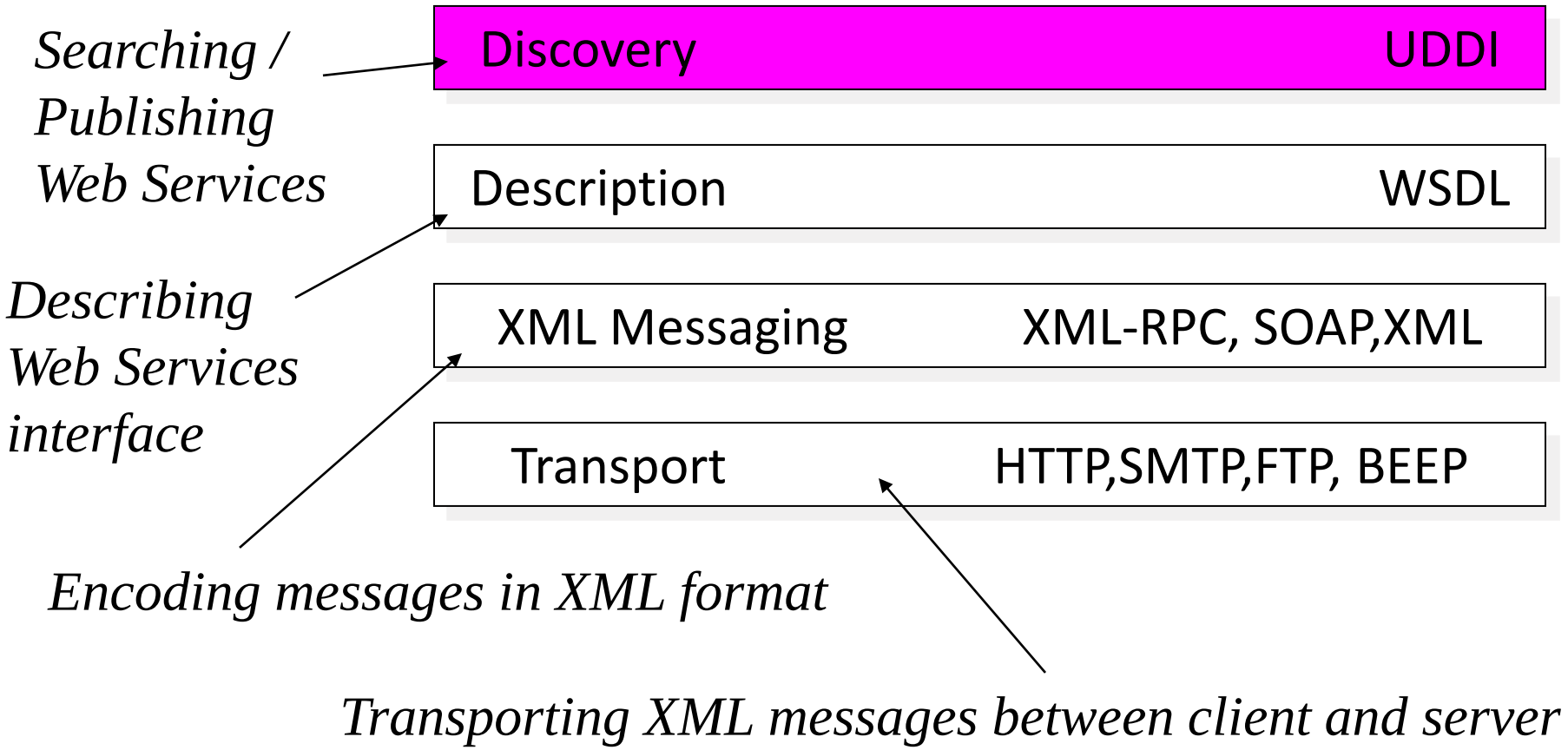
Module Three: Service Discovery UDDI and Service Composition BPEL

Module 3 Guiding Questions

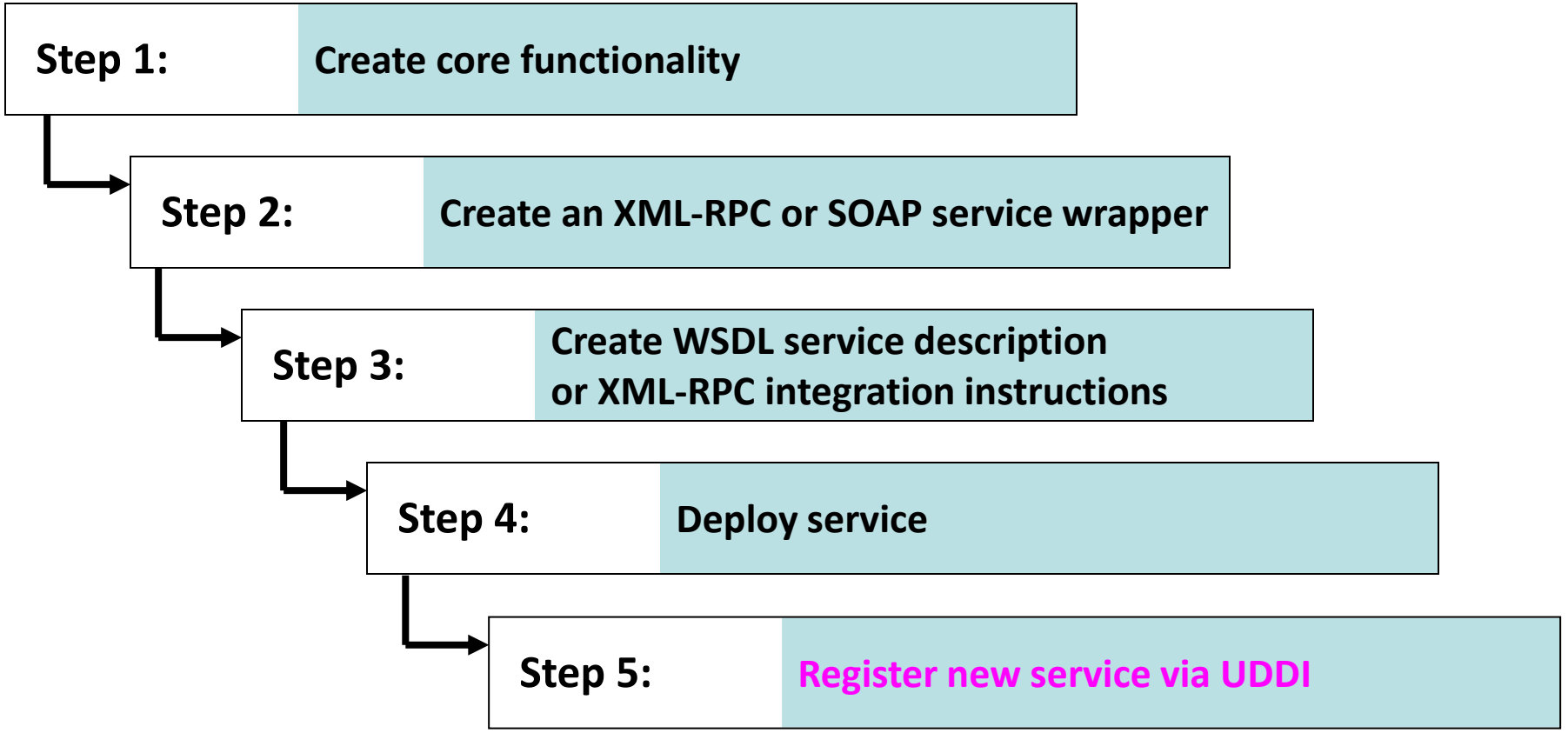
- What is service discovery?
- What is UDDI?
- What are the main uses of UDDI?
- What is service composition?
- What is business process?

Service Discovery UDDI

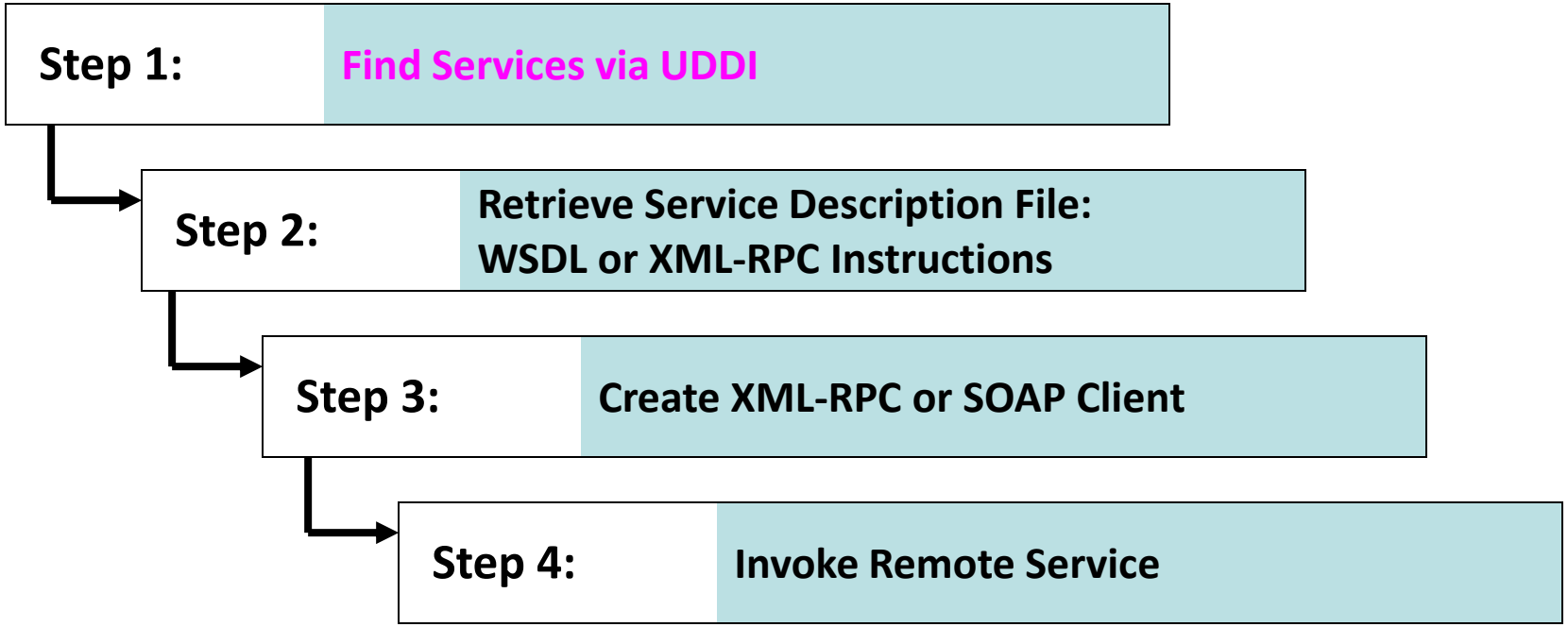
Web Service Protocol Stack



Using the Protocols Together – service provider perspective



Using the Protocols Together – service request perspective



UDDI implementations

- Public federated network of UDDI registries available on the Internet
  - UDDI registry is a part of the global federated network
- Private UDDI registries developed by companies or industry groups
  - UDDI registry is privately owned and operated
  - Allows members of the company or the industry group to share and advertise services amongst themselves

- In common: Web services API for publishing and locating businesses and services advertised within the UDDI registry

UDDI current use – private implementations

- Support SOA governance within an organization
- Offering a standardized way to
  - catalog company matadata
  - categorize it in multiple dimensions
- Used to dynamically bind client systems to implementations

Service Composition BPEL

What is service composition?

- Service composition allows developers to “compose” services that exchange SOAP messages and define their interfaces into an aggregate solution.
- The aggregate is a composed Web service or a so-called composite Web service.

What is BPEL?

- The Business Process Execution Language for Web Services is such a flow representation developed to facilitate coordination of Web services into a comprehensive business process.
- In short: BPEL is a business process language that can be used to represent composite services.

Module Four: Basics of web service programming; developing web services with JEE platform

Module 4 Guiding Questions

- What is Java EE?
- What is JAX-WS?
- What technologies do you know for developing web services?

Java EE – Java Enterprise Edition

- Java Platform, Enterprise Edition
- Formerly known as
  - Java 2 Platform, Enterprise Edition (J2EE)
- The current version is known as
  - Jakarta EE
- A set of specifications, extending Java SE 8 with specifications for enterprise features such as distributed computing and web services
- Java EE is defined by its specification
  - It defines APIs and their interactions

Java EE specifications

- Web specifications
  - Servlet
  - WebSocket
  - Java Server Faces
  - Unified Expression Language
- Web Service specifications
  - Java API for RESTFful Web Services
  - Java API for JSON Processing
  - JAVA API for JSON Binding
  - Java Architecture for XML Binding
  - Java API for XML Web Services
- Enterprise specifications
  - Contexts and Dependency Injection
  - Enterprise JavaBean
  - Java Persistence API
  - Java Transaction API
  - Java Message Service
- Other specifications
  - Validation
  - Batch Applications
  - Java EE Connector Architecture

JAX-WS

- A part of the Java EE platform
- Java API for XML Web Services
- A Java programming language API for creating web services, particularly SOAP services
- One of the JAVA XML programming APIs
- It can be used in Java SE starting with version 6

JAX-WS 2.2 specification

- https://jcp.org/en/jsr/detail?id=224
- Defines a standard Java-to-WSDL mapping which determines how WSDL operations are bound to Java methods when a SOAP message invokes a WSDL operation
  - Determines which Java method gets invoked and how that SOAP message is mapped to the method's parameters
  - Determines how the method's return value gets mapped to the SOAP response

Module Five: Service Oriented Architecture: architectural patterns and modelling methods of SOA, designing a distributed service system with SOA

Guided questions for Module 5

- What is a service oriented architecture?
- What are services within SOA?
- What are the principles of service-orientation? Do you understand then in details?
- What are characteristics of contemporary SOA?
- What are the benefits of SOA?
- What are the pitfalls/challenges of adopting SOA? When not to use SOA?
- How service orientation principles inter-relate?
- How Web services support service orientation principles?
- What is Enterprise Service Bus?

Service Oriented Architecture

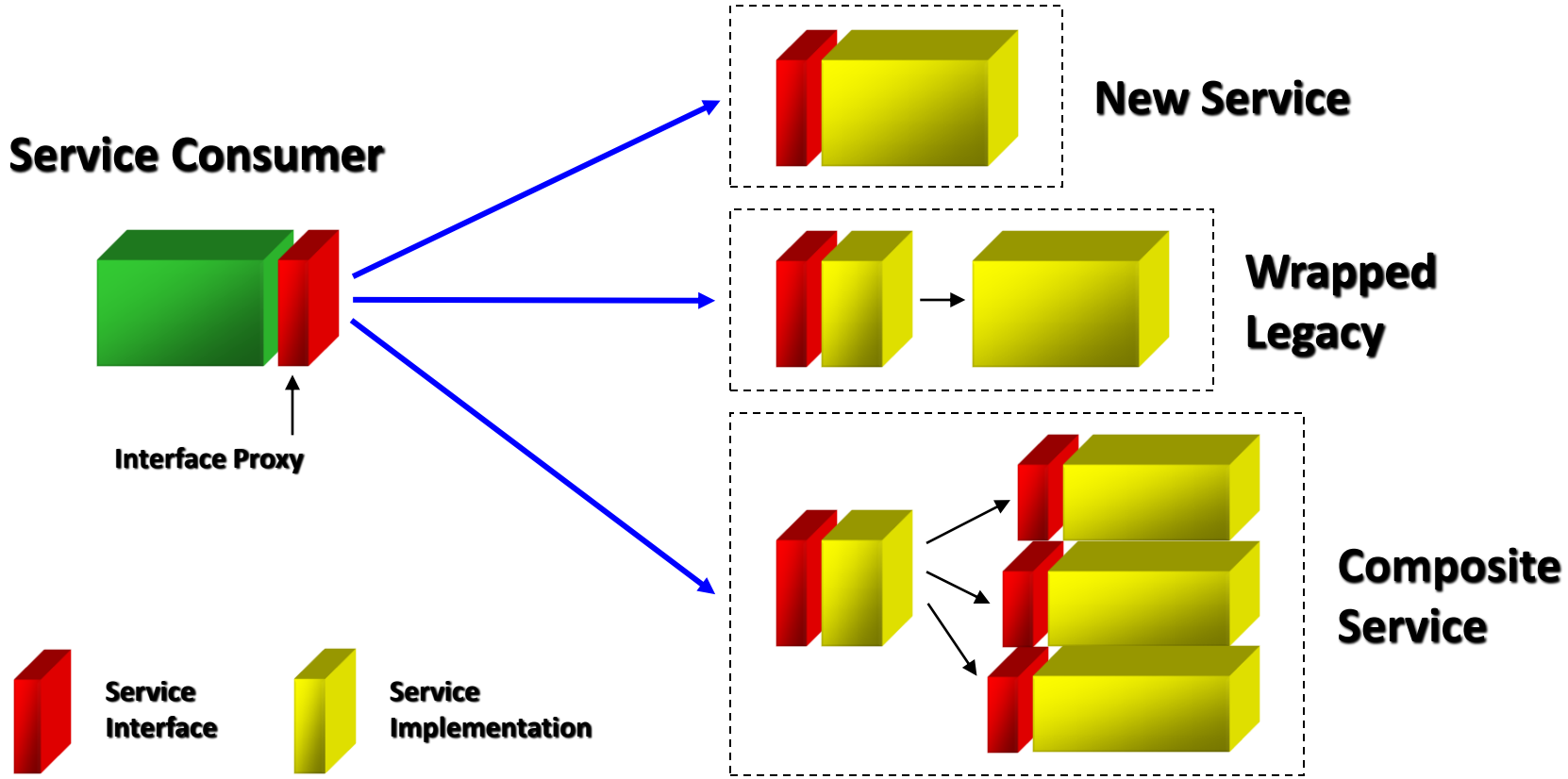
What is SOA?

- Service-Oriented Architecture (SOA) is a set of principles and methodologies for designing and developing software in the form of interoperable services.
- These services are well-defined business functionalities that are built as software components that can be reused for different purposes.
- SOA design principles are used during the phases of systems development and integration.

SOA Service

- A discrete unit of functionality that can be accessed remotely and acted upon and updated independently
- A repeatable task, for example:
  - Open an account
  - Perform a credit check
  - retrieving a credit card statement online

Anatomy of a Service

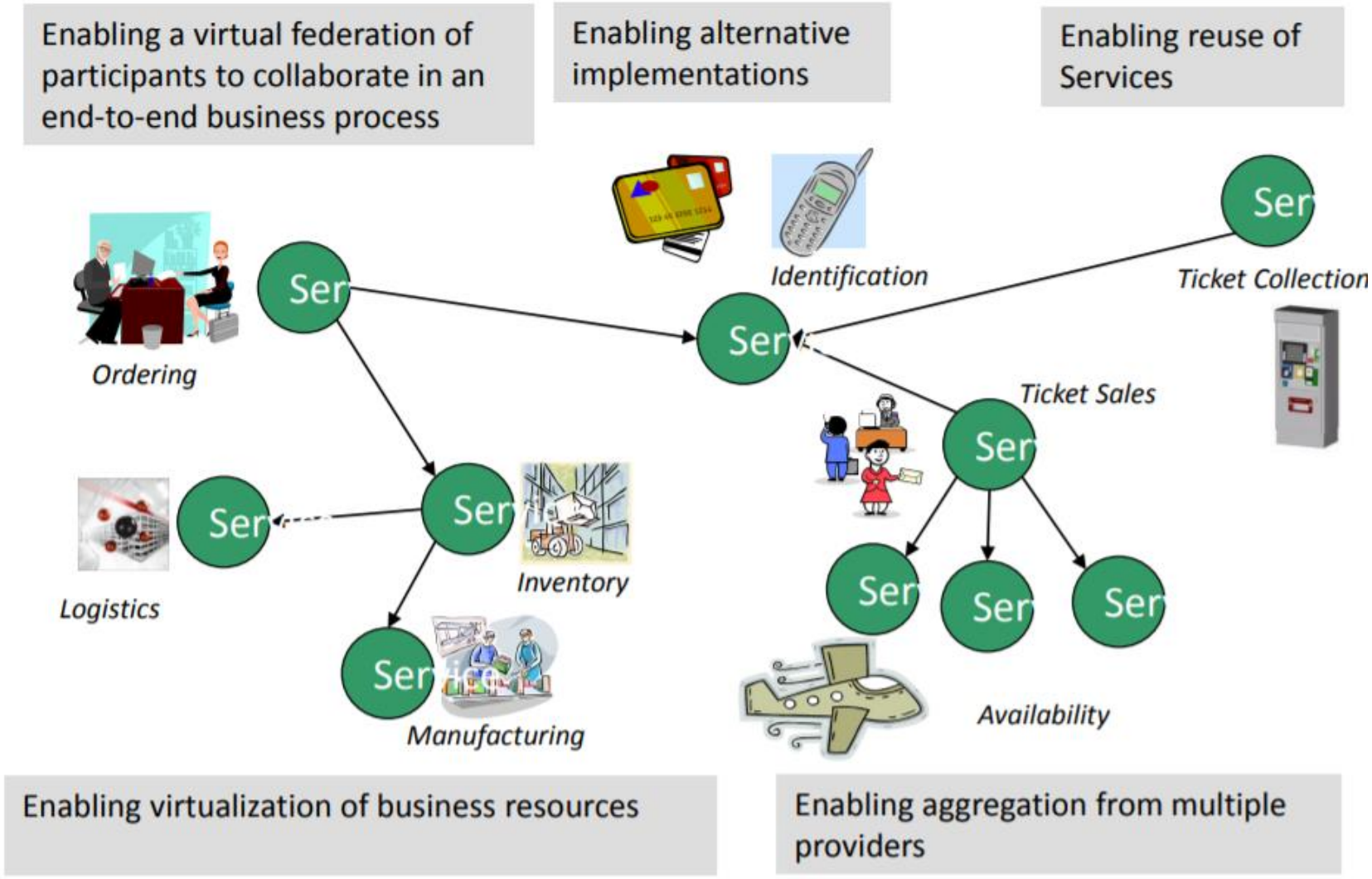


Why SOA in Enterprise?

- Why to introduce SOA in an organization? What are the benefits for enterprises?
- Can you make a recommendation to use SOA in a particular case of an enterprise by analyzing their business/technical problems and pointing out the relevant possible benefits?

Why SOA in Enterprises? ENABLE FLEXIBLE, FEDERATED BUSINESS PROCESSES

- Enable flexible, federated business processes



Business benefits

- Decreased cost:
  - Add value to core investments by leveraging existing assets
  - New systems can be built faster for less money
    - Reducing integration expense
    - Built for flexibility
    - Long term value of interoperability
- Increased employee productivity:
  - Built on existing skills
  - Consolidate duplicate functionality

Business benefits

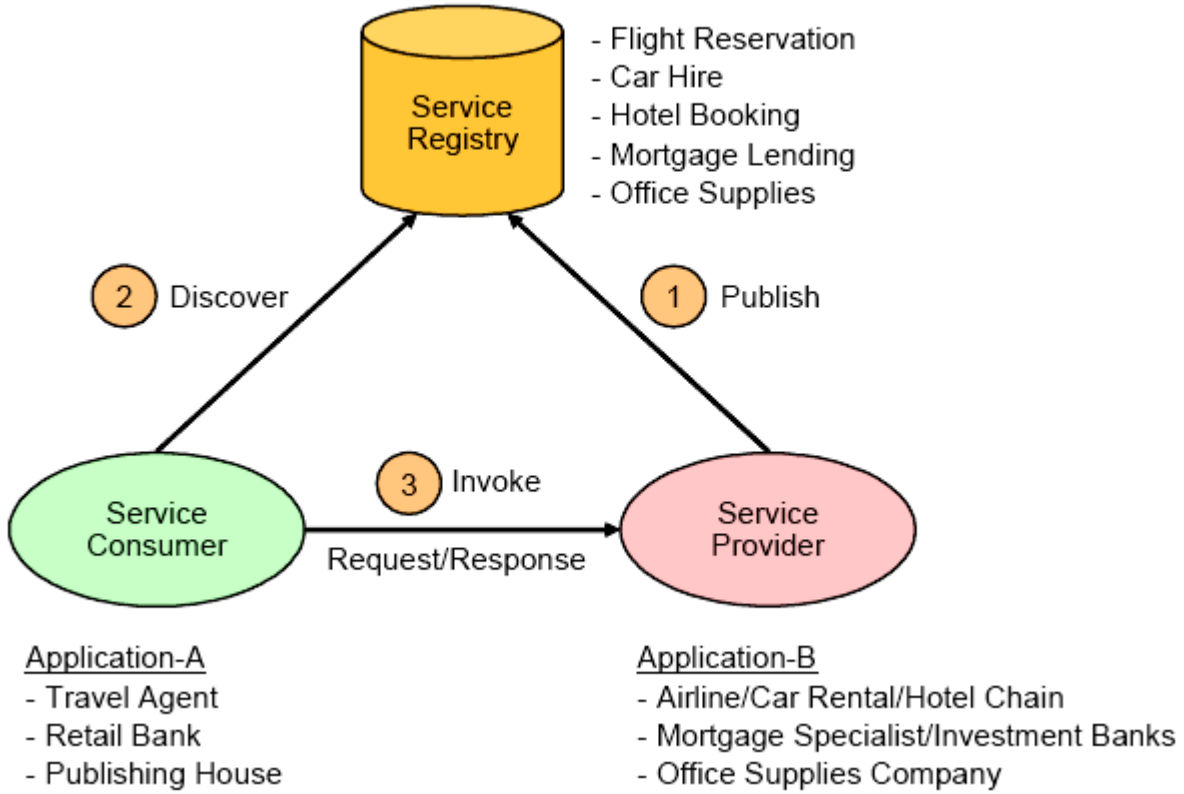
- Built for partnerships:
  - Standards based
  - Business relationships expressed via service interactions
  - Integration is driven by what is needed, not what is technically possible
- Agility - Built for change
  - Helps applications evolve over time and last
  - Abstract the backend and replace over time
  - Focusing on core-competencies
  - Incremental implementation approach is supported
  - Service Outsourcing – new business model!

Technical Benefits

- Services Scale
  - Build scalable, evolvable systems
  - Scale down to mobile devices
  - Scale up to for large systems or across organizations
- Manage complex systems
  - Does not require centralized services
  - Empowers users with high end communication
- Platform independent use
- Loose Coupling allows flexibility

SOA Components and Operations

- Can you identify in a specific example which parts of a SOA solution are service consumers, which are service providers?



Characteristics of service-oriented architectures

- Any function can be a service but need not be. The criteria for whether a function needs to be a service is based on its reuse potential
  - Business process services
    - createStockOrder, reconcileAccount, renewPolicy
  - Business transaction services
    - checkOrderAvailability, createBillingRecord
  - Business function services
    - calculateDollarValueFromYen, getStockPrice
  - Technical function services
    - auditEvent, checkUserPassword,checkUserAuthorisation

Service Design Principles

- The principles of service-orientation provide a means of supporting and achieving a foundation paradigm based upon building of SOA characteristics.

Common Service Design Principles

- Standardized Service Contracts
- Loose Coupling
- Abstraction
- Reusability
- Autonomy
- Statelessness
- Discoverability
- Composability

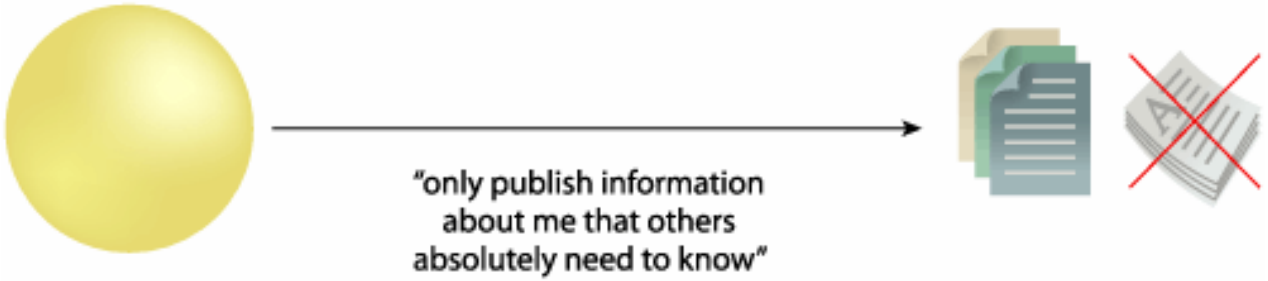
Standardized Service Contracts

- Services share a formal contract
- Services adhere to a communications agreement as defined collectively by one or more service description documents (for example WSDL in case of Web Services)
- For services to interact, they need not share anything but a formal contract that describes each service and defines the terms of information exchange.

Loose Coupling

- Services are loosely coupled
- Services maintain a relationship that minimizes dependencies and only maintain an awareness of each other
- Services must be designed to interact without the need for tight, cross-service dependencies
- A service is defined solely by an implementation-independent interface
- Services should be able to change their implementation without impacting service consumers

Abstraction



- Services abstract underlying logic
- Beyond what is described in the service contract, services hide logic from the outside world
- The only part of a service that is visible to the outside world is what is exposed via the service contract. Underlying logic, beyond what is expressed in the descriptions that comprise the contract, is invisible and irrelevant to service requestors.

Reusability

- Services are reusable
- Logic is divided into services with the intention of promoting reuse
- Regardless of whether immediate reuse opportunities exist; services are designed to support potential reuse

Autonomy

- Services are autonomous
- Services have control over the logic they encapsulate
- The logic governed by a service resides within an explicit boundary. The service has control within this boundary and is not dependent on other services for it to execute its governance.

Statelessness

- Services are stateless
- Services should not be required to manage state information, as that can impede their ability to remain loosely coupled. Services should be designed to maximize statelessness even if that means deferring state management elsewhere.
- Service implementations should not hold conversational state across multiple requests.
  - Communicate complete information at each request.
- Each operation should be functionally isolated (separate, independent).

Discoverability

- Services are discoverable
- Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms (for example UDDI incase of Web services)
- Services should allow their descriptions to be discovered and understood by humans and service requestors that may be able to make use of their logic.

Composability

- Services are composable
- Collections of services can be coordinated and assembled to form composite services
- Services may compose other services. This allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers.

Module Six: Service Engineering

Guided questions for Module 6

- What is a series of steps that need to be completed to construct the services for a given service-oriented solution?
- What is a lifecycle of an SOA delivery project?
- What are SOA delivery lifecycle phases?
- What is the purpose of service-oriented analysis? What are the steps taken in this phase?
- What happens in service-oriented design phase of an SOA delivery lifecycle?
- What choices are made during service development?

Guided questions for Module 6

- What are three common SOA delivery strategies?
- What is the process of top-down strategy?
- What are the advantages and disadvantages of top-down strategy?
- What is the process of bottom-up strategy?
- What are the advantages and disadvantages of bottom-up strategy?
- What is the process of agile strategy?
- What are the advantages and disadvantages of agile strategy?

SOA delivery strategies

- Different strategies exist for how to organize lifecycle stages to enable delivery of specialized service layers
  - Top-down
  - Bottom-up
  - Agile (meet-in-the-middle)

Top-down strategy

- "analysis first" approach
- Promotes the formal definition of corporate business models prior to modeling service boundaries

Top-down strategy benefits and weaknesses

- It can result in highest quality level of SOA
- Significant volume of up-front analysis work

Bottom-up strategy

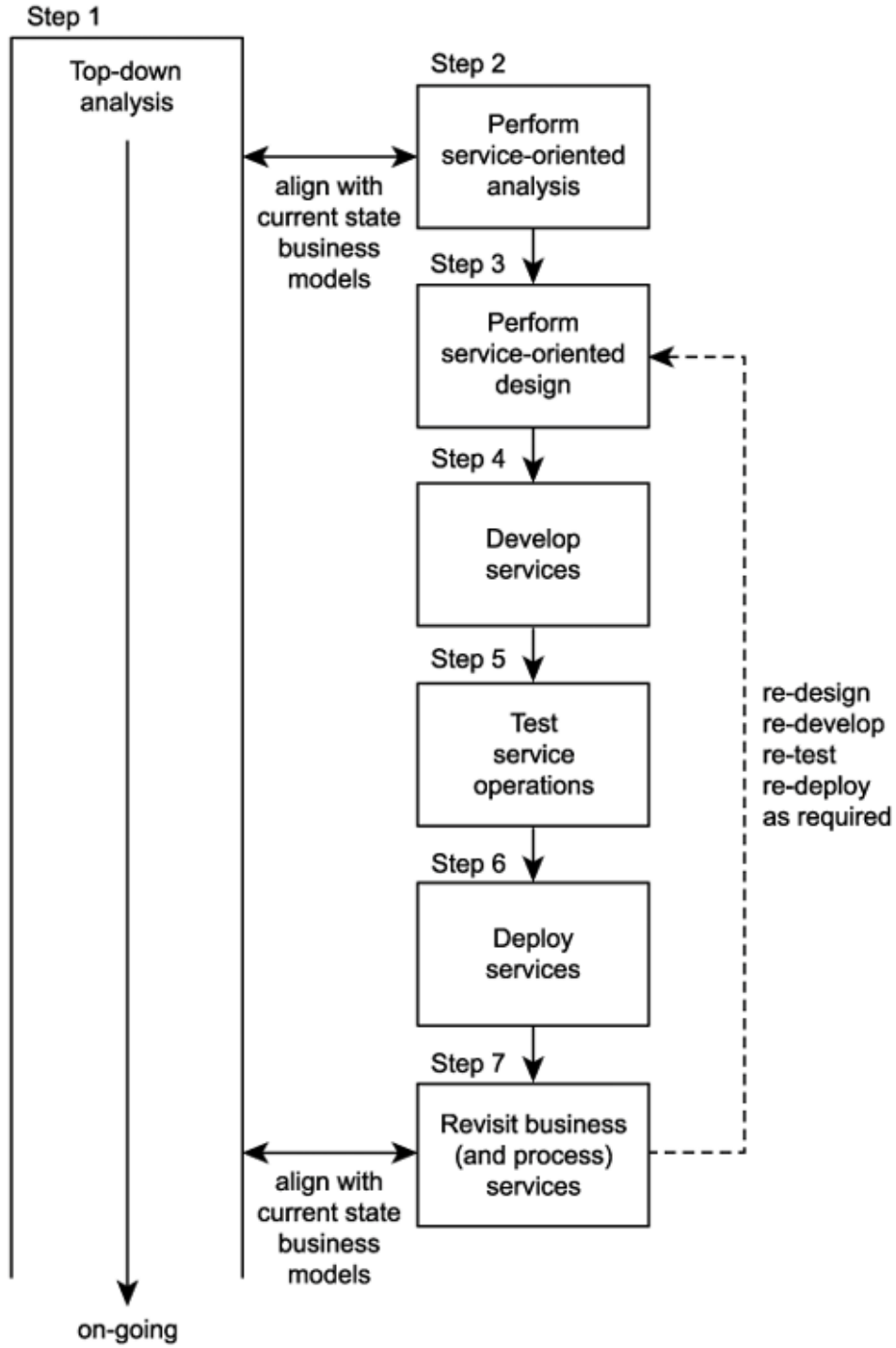
- This approach encourages the creation of services as a means of fulfilling application-centric requirements.
- Based on the delivery of application services on an “as needed” basis
- Integration is the primary motivator for bottom-up designs, where the need to take advantage of the open SOAP communications framework can be met by simply appending services as wrappers to legacy systems.

Bottom-up strategy benefits and weaknesses

- Easy to follow
- Does not result in the advancement of service-orientation

Agile strategy

- A combination of top-down and bottom-up approaches
- Business-level analysis occurs concurrently with service design and development


Step 1
Top-down analysis
Step 2
Perform service-oriented analysis
align with current state business models
Step 3
Perform service-oriented design
Step 4
Develop services
Step 5
Test service operations
Step 6
Deploy services
Step 7
Revisit business (and process) services
re-design
re-develop
re-test
re-deploy
as required
align with current state business models
on-going

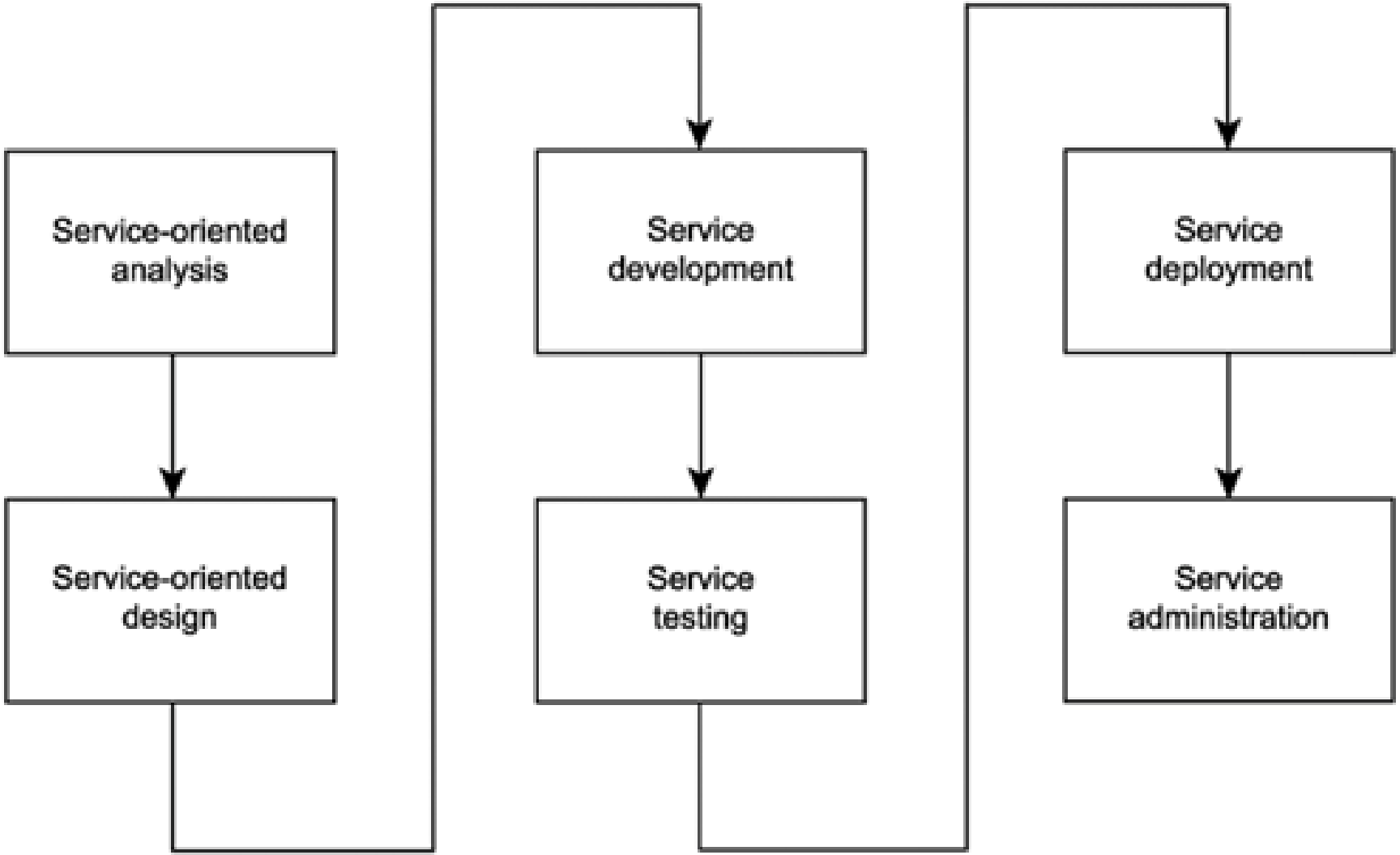
Agile strategy benefits and weaknesses

- On-going analysis is supported, while allowing immediate delivery of service
- As analysis progresses, existing services are revisited and revised as required
- More complex - it needs to fulfill two opposing sets of requirements
- Increased effort associated with the delivery of every service
  - services may need to be revisited, redesigned, redeveloped, and redeployed

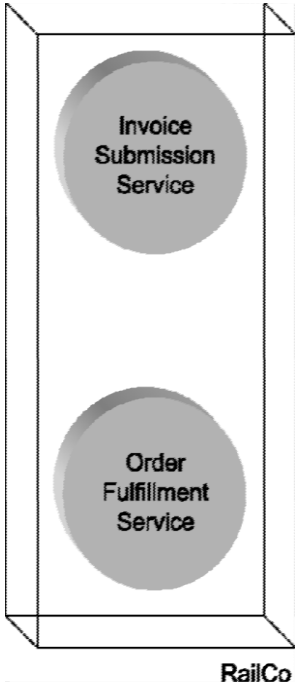
Discussion

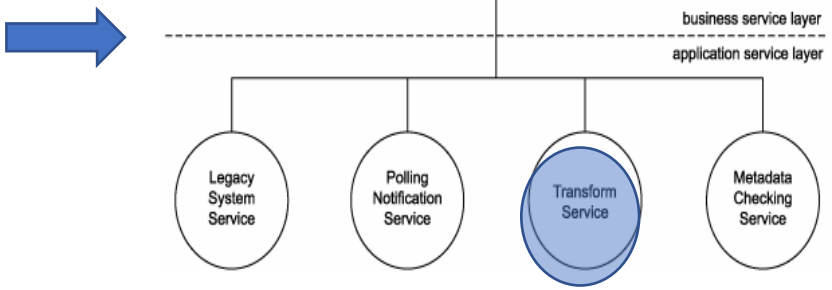
- What challenges and benefits do you see in redeveloping the Web services constructed based on:
  - bottom-up approach
  - top-down approach
  - agile approach
- Can you make recommendation based on the context of the specific enterprise?
  - Review the relevant ppt slides to practice

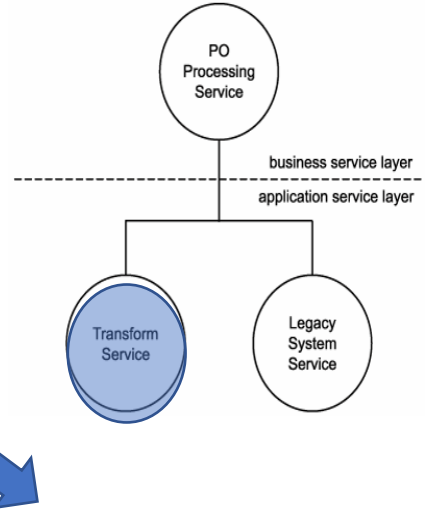
Common phases of an SOA delivery lifecycle

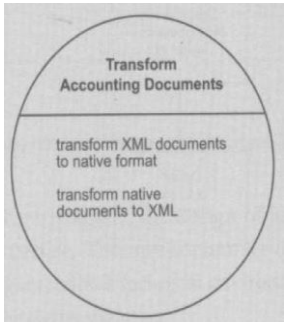


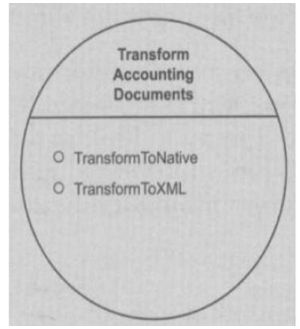
SOA in an Enterprise

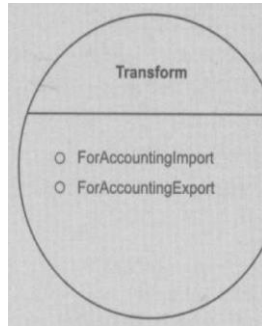






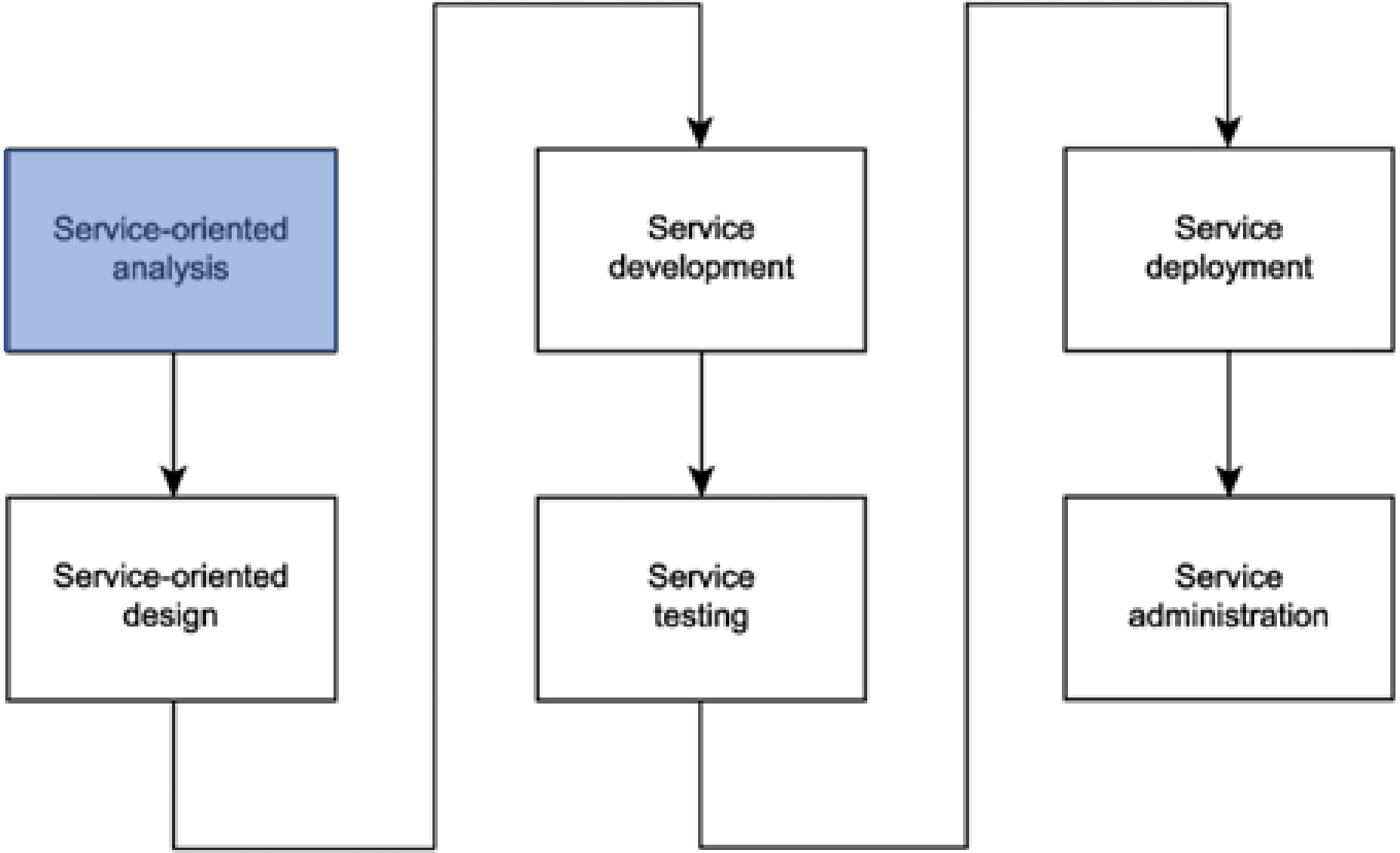


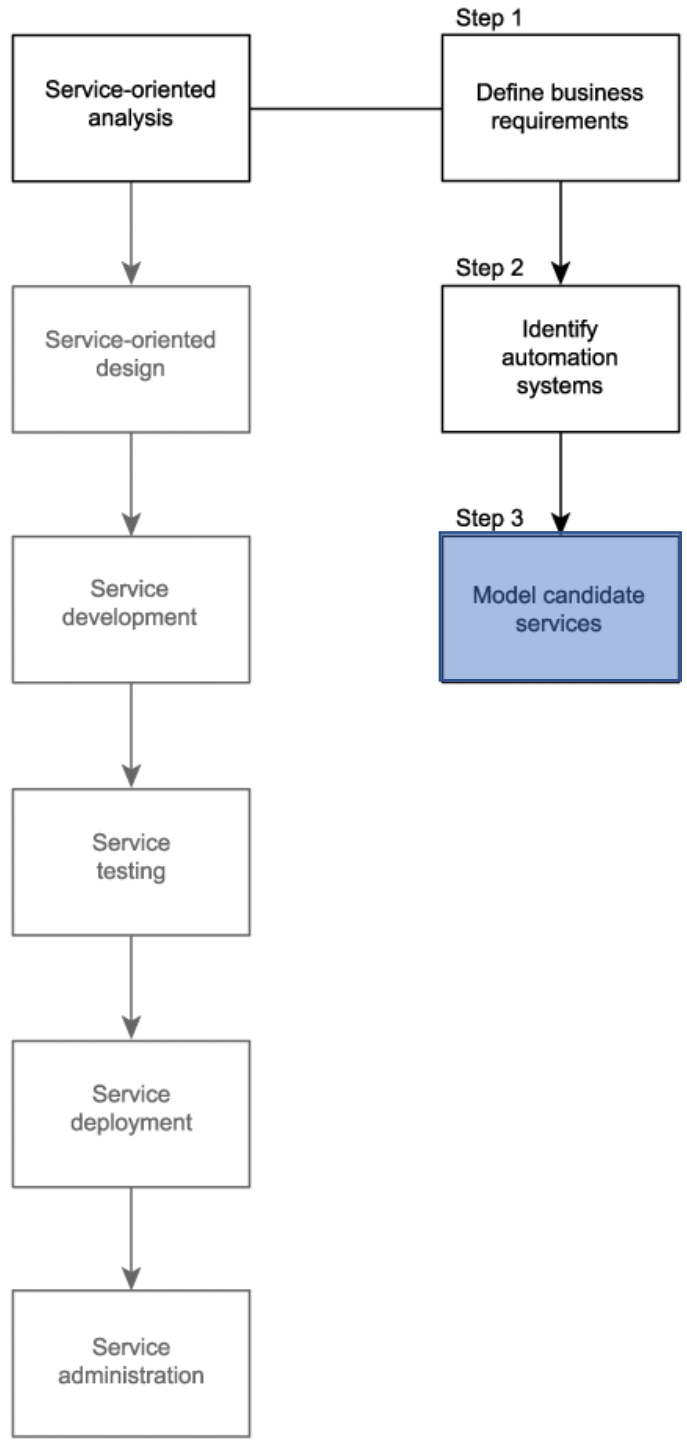




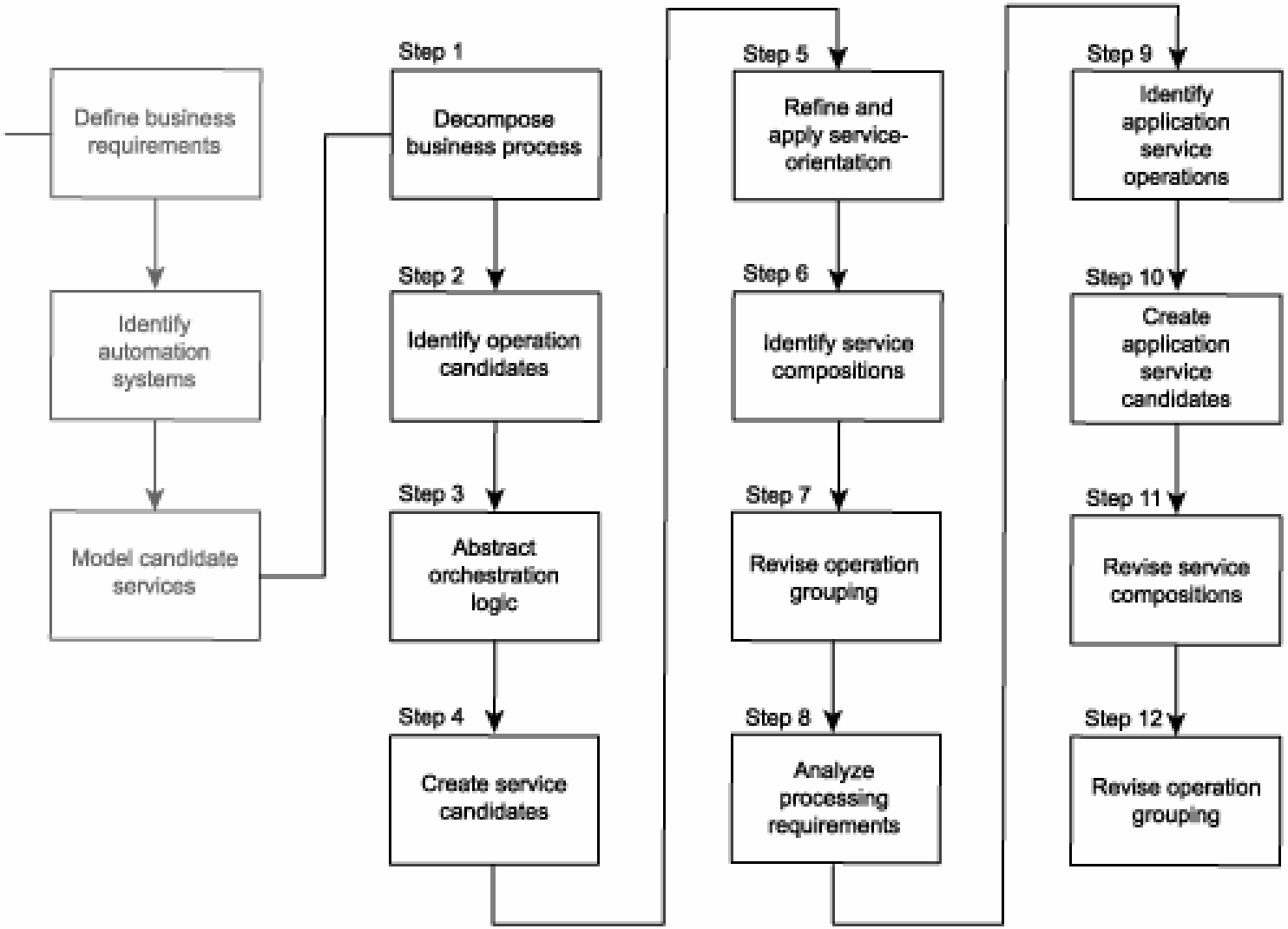
- Make sure to review the ppt slides from the lectures where we described this process in details and explained why these decisions were made
- The following slides will only review the results of each step

Service-oriented analysis review




Step 1
Service-oriented analysis
Define business requirements
Step 2
Service-oriented design
Identify automation systems
Step 3
Service development
Model candidate services
Service testing
Service deployment
Service administration

Common steps of modelling process

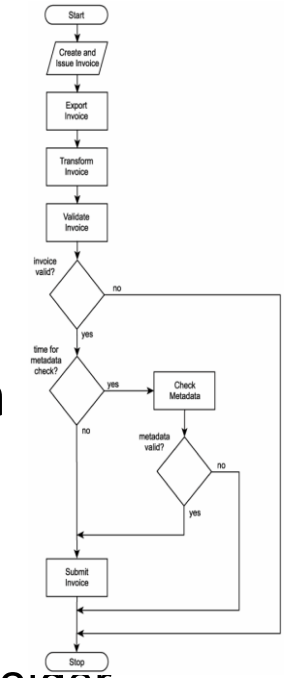


- Are you able to perform the service analysis steps for the example scenario?

From business process to decomposed business process

Invoice Submission Process

- Accounting clerk creates and issues an electronic invoice using the legacy accounting system.
- The save event triggers a custom script that exports an electronic copy of the invoice to a network folder.
- A custom developed component, which polls this folder at ten-minute intervals, picks up the document and transforms it into an XML document.
- The invoice XML document is then validated. If it is deemed valid, it is forwarded to the Invoice Submission Service. If validation fails, the document is rejected, and the process ends.
- Depending on when the last metadata check was performed, the service may issue a Get Metadata request to the TLS B2B solution.
- If the Get Metadata request is issued and if it determines that no changes were made to the relevant TLS service descriptions, the Invoice Submission Service transmits the invoice document to the TLS B2B solution using the ExactlyOnce delivery assurance. If the Get Metadata request identifies a change to the TLS service descriptions, the invoice is not submitted, and the process ends.



Decomposed Invoice Submission Process

- Create electronic invoice.
- Issue electronic invoice.
- Export electronic invoice to network folder.
- Poll network folder.
- Retrieve electronic invoice.
- Transform electronic invoice to XML document.
- Check validity of invoice document. If invalid, end process.
- Check if it is time to verify TLS metadata.
- If required, perform metadata check. If metadata check fails, end process.

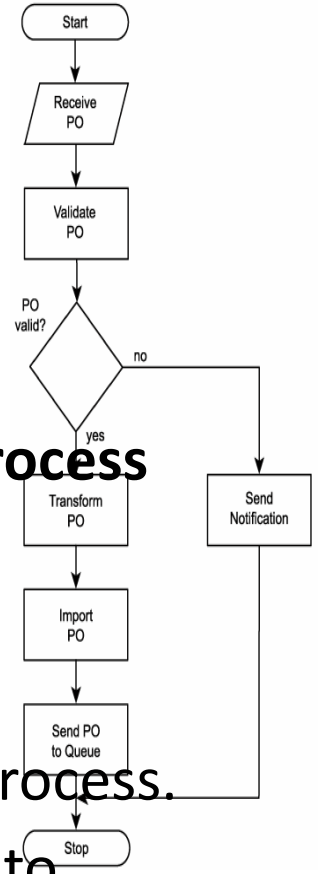
From business process to decomposed business process

Order Fulfilment Process

- The RailCo Order Fulfilment Service receives a SOAP message from TLS, containing a payload consisting of a TLS purchase order document.
- The service validates the incoming document. If valid, the document is passed to a custom component. If the TLS PO fails validation, a rejection notification message is sent to TLS, and the process ends.
- The component has the XML document transformed into a purchase order that conforms to the accounting system's native document format.
- The PO then is submitted to the accounting system using its import extension.
- The PO ends up in the work queue of an accounting clerk who then processes the document.

Decomposed Order Fulfilment Process

- Receive PO document.
- Validate PO document.
- If PO document is invalid, send rejection notification and end process.
- Transform PO XML document into native electronic PO format.
- Import electronic PO into accounting system.
- Send PO to accounting clerk's work queue.



From decomposed business process to service operation candidates

Decomposed Invoice Submission Process

- Create electronic invoice.
- Issue electronic invoice.
- Export electronic invoice to network folder.
- Poll network folder.
- Retrieve electronic invoice.
- Transform electronic invoice to XML document.
- Check validity of invoice document. If invalid, end process.
- Check if it is time to verify TLS metadata.
- If required, perform metadata check. If metadata check fails, end process.

Invoice Submission operation candidates:

- ~~Create electronic invoice.~~ (A manual step performed by the accounting clerk.)
- ~~Issue electronic invoice.~~ (A manual step performed by the accounting clerk.)
- Export electronic invoice to network folder. (Currently a custom developed extension of the legacy system. Could be made part of a generic service candidate.)
- Poll network folder. (Currently performed by a custom developed component. Could be made part of a service candidate.)
- Retrieve electronic invoice. (Same as previous.)
- Transform electronic invoice to XML document. (Same as previous.)
- Check validity of invoice document. If invalid, end process. (Is currently being performed as part of the Invoice Submission Service's parsing routine. No foreseeable need to change this.)
- Check if it is time to verify TLS metadata. (Is currently being performed as part of the Invoice Submission Service's parsing routine. Looks like a potentially reusable operation candidate. Could be moved to a separate service candidate.)
- If required, perform metadata check. If metadata check fails, end process. (Same as previous.)

From decomposed business process to service operation candidates

Decomposed Order Fulfilment Process

- Receive PO document.
- Validate PO document.
- If PO document is invalid, send rejection notification and end process.
- Transform PO XML document into native electronic PO format.
- Import electronic PO into accounting system.
- Send PO to accounting clerk's work queue.

Order Fulfillment operation candidates

- Receive PO document. (Is currently being performed by the Order Fulfillment Service. No foreseeable need to change this.)
- Validate PO document. (Same as previous.)
- If PO document is invalid, send rejection notification and end process. (Same as previous.)
- Transform PO XML document into native electronic PO format. (Currently performed by a custom developed component. Could be made part of a service candidate.)
- Import electronic PO into accounting system. (Currently a custom developed extension of the legacy system. Could be made part of a generic service candidate.)
- Send PO to accounting clerk's work queue. (Same as previous.)

From service operation candidates to service candidates

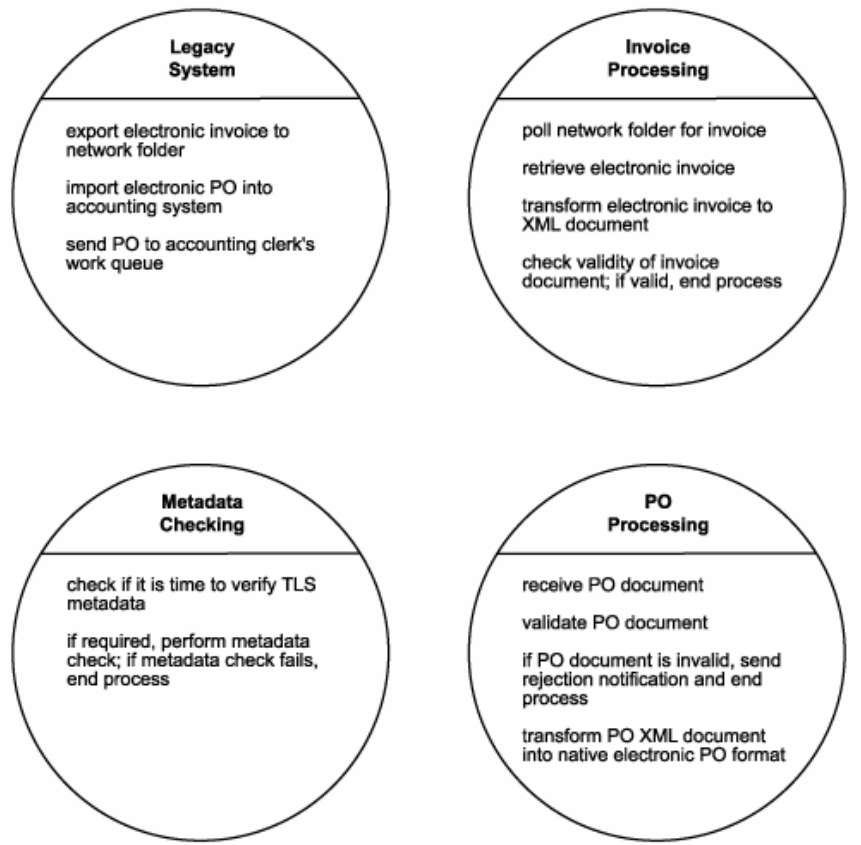
Invoice Submission operation candidates:

- ~~Create electronic invoice.~~ (A manual step performed by the accounting clerk.)
- ~~Issue electronic invoice.~~ (A manual step performed by the accounting clerk.)
- Export electronic invoice to network folder. (Currently a custom developed extension of the legacy system. Could be made part of a generic service candidate.)
- Poll network folder. (Currently performed by a custom developed component. Could be made part of a service candidate.)
- Retrieve electronic invoice. (Same as previous.)
- Transform electronic invoice to XML document. (Same as previous.)
- Check validity of invoice document. If invalid, end process. (Is currently being performed as part of the Invoice Submission Service's parsing routine. No foreseeable need to change this.)
- Check if it is time to verify TLS metadata. (Is currently being performed as part of the Invoice Submission Service's parsing routine. Looks like a potentially reusable operation candidate. Could be moved to a separate service candidate.)
- If required, perform metadata check. If metadata check fails, end process. (Same as previous.)

Order Fulfillment operation candidates:

- Receive PO document. (Is currently being performed by the Order Fulfillment Service. No foreseeable need to change this.)
- Validate PO document. (Same as previous.)
- If PO document is invalid, send rejection notification and end process. (Same as previous.)
- Transform PO XML document into native electronic PO format. (Currently performed by a custom developed component. Could be made part of a service candidate.)
- Import electronic PO into accounting system. (Currently a custom developed extension of the legacy system. Could be made part of a generic service candidate.)
- Send PO to accounting clerk's work queue. (Same as previous.)

service candidates



How did we refine our service candidates?

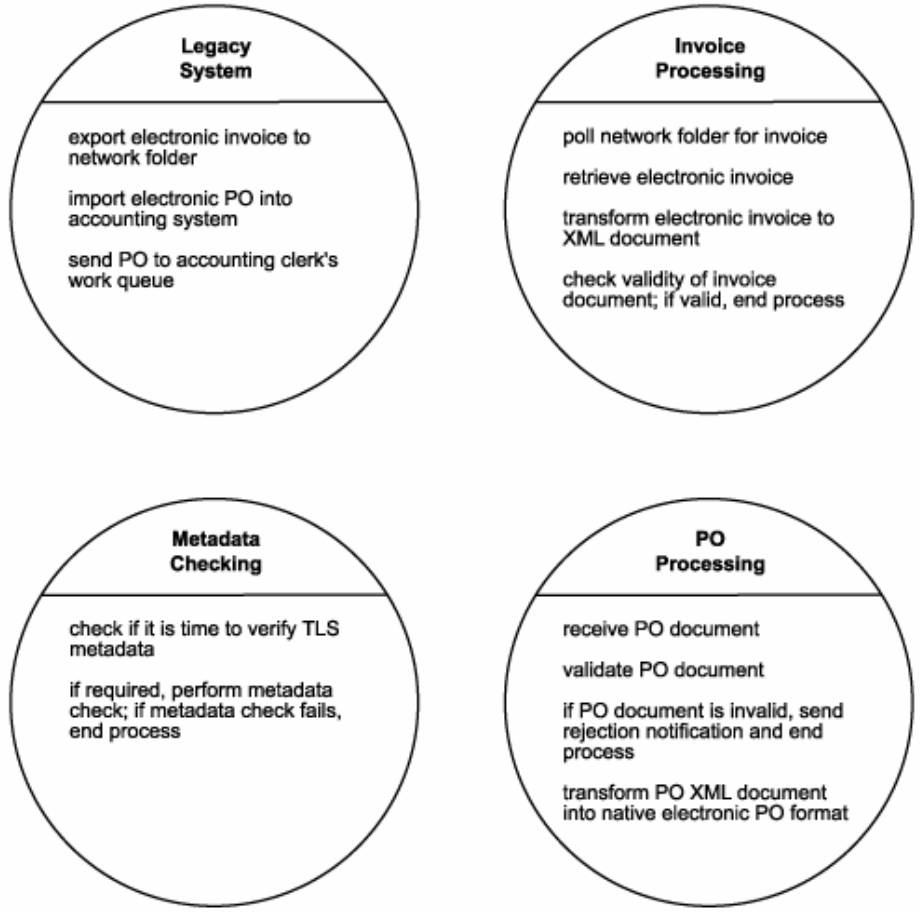
- Do you know how to perform each step? How to apply the service orientation principles? How to make the adjustments?

Step 5: Refine and apply principles of service-orientation

- So far we have just grouped processing steps derived from an existing business process
- To make our service candidates truly worthy of an SOA, we must take a closer look at the underlying logic of each proposed service operation candidate
- In this step we will make adjustments by applying key service-orientation principles
  - reusability
  - autonomy

Step 5 RailCo example (1)

- We will continue our RailCo case to illustrate this step
- Let us recall the initial service candidates we identified in step 4
- We will review the operation candidates within our service candidates and make a series of adjustments



How did we refine our service candidates?

- Do you know how to apply reusability and autonomy principles to adjust the service candidates in our initial design?

Step 5 RailCo example (2)

- Within the Legacy System Service, the "Send PO to accounting clerk's work queue" action can be performed only upon the receipt of a document. This operation candidate is therefore directly dependent on the "Import electronic PO into accounting system" step. We therefore decide to combine these two steps into one.
- Adjustment:
  - A new operation candidate "Import and forward document to work queue" is a combination of:
    - "Send PO to accounting clerk's work queue"
    - "Import electronic PO into accounting system“

Step 5 RailCo example (3)

- Further the "Export electronic invoice to network folder" action is performed automatically by a macro added to the legacy accounting system. It is therefore not required as part of our service candidate. This leaves us with a single operation candidate that we would like to make more reusable by allowing it to handle different types of documents.
- Adjustment
  - Make “Export electronic invoice to network folder” more reusable by designing it in more generic way
    - “export document to network folder”

Step 5 RailCo example (4)

- The revised Legacy System Service list contains the following steps:
  - Export document to network folder.
  - Import and forward document to work queue.

- Let’s now examine Invoice Processing Service
- One of operation candidates is "Poll network folder for invoice"
- Let’s apply the principle of **reusability**
- How can we make this operation more reusable?

Step 5 RailCo example (5)

- Upon reviewing the Invoice Processing Service, a number of refinement opportunities arise. We determine that the "Poll network folder for invoice" action can be made more generic by turning it into an operation candidate that simply polls a given folder for different types of documents.
- Adjustment
  - "Poll network folder for invoice“ becomes “poll folder for new documents”

Step 5 RailCo example (6)

- We also decide that this action should be made part of a service candidate capable of notifying subscribers of the arrival of new documents.
- The result of our analysis is a new context (a new service candidate), established to represent generic notification actions, as follows:
  - Polling Notification Service:
    - Poll folder for new documents.
    - If documents arrive for which there are subscribers, issue notifications.

Step 5 RailCo example (7)

- Next, we decide to combine the "Retrieve electronic invoice," "Transform electronic invoice to XML document," and "Check validity of invoice document" operation candidates into a single service operation candidate called "Retrieve and transform invoice document."
- We don't mention the validation aspect of this action because the XML document automatically is assigned a corresponding schema. The validation of the document is therefore an intrinsic part of the transformation process.
- The revised Invoice Processing Service list is left with just one step:
  - Retrieve and transform invoice document.

Step 5 RailCo example (8)

- Next, we tackle the operation candidates in the PO processing group. Though listed as such, the "Receive PO document" is not a suitable service operation candidate, as receiving a message is a natural part of service operations (and therefore not generally explicitly accounted for).
- Adjustment
  - Remove "~~Receive PO document~~" from our list.

Step 5 RailCo example (9)

- We then detect a direct dependency between the "If PO document is invalid, send rejection notification and end process" and "Validate PO document" actions.
- Adjustments
  - As a result we decide to combine these into a single operation candidate called "Validate PO document and send rejection notification, if required.“

Step 5 RailCo example (10)

- We move on to discover commonality between the "Transform PO XML document into native electronic PO format" action and the "Retrieve and transform invoice document" action from our Invoice Processing Service list.
- Both operation candidates transform accounting documents. We therefore decide to create a new service candidate that provides generic transformation.
- Adjustment - The result is a new grouping category:
  - Transform Accounting Documents Service:
    - Transform XML documents to native format.
    - Transform native documents to XML.

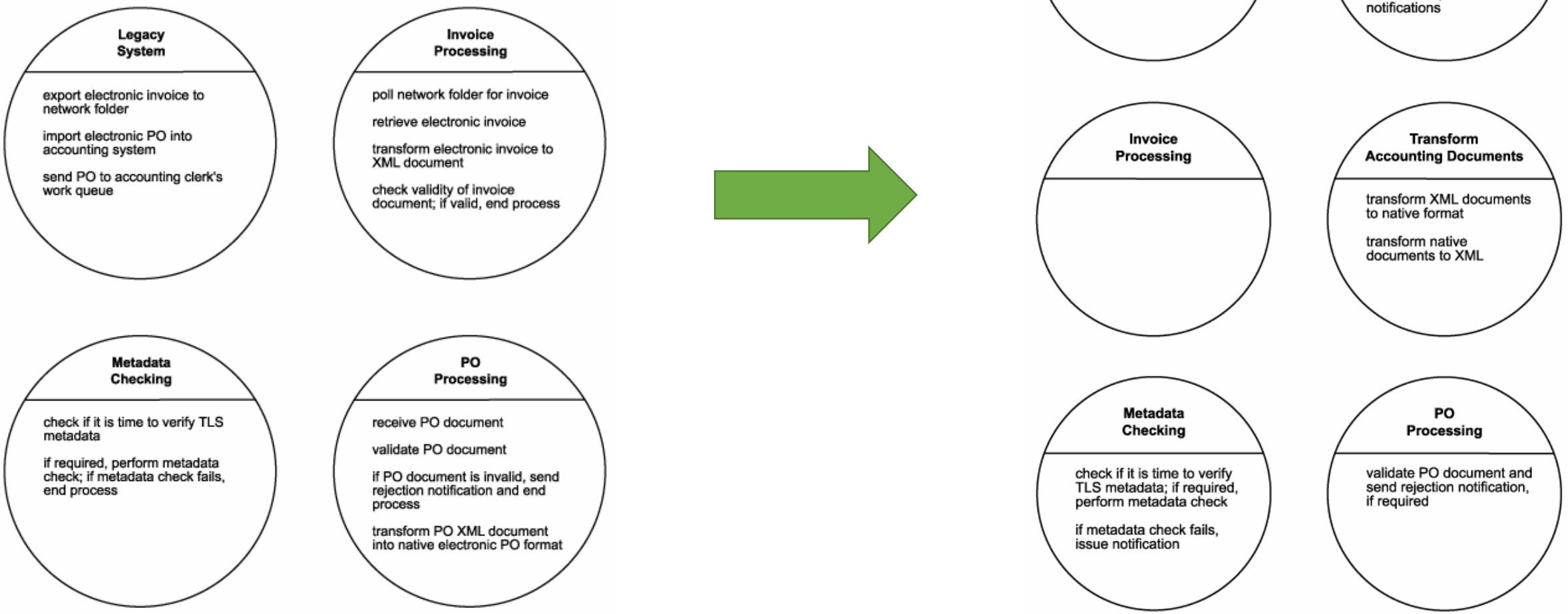
Step 5 RailCo example (11)

- The revised PO Processing Service list is left with just one step:
  - Validate PO document and send rejection notification, if required.

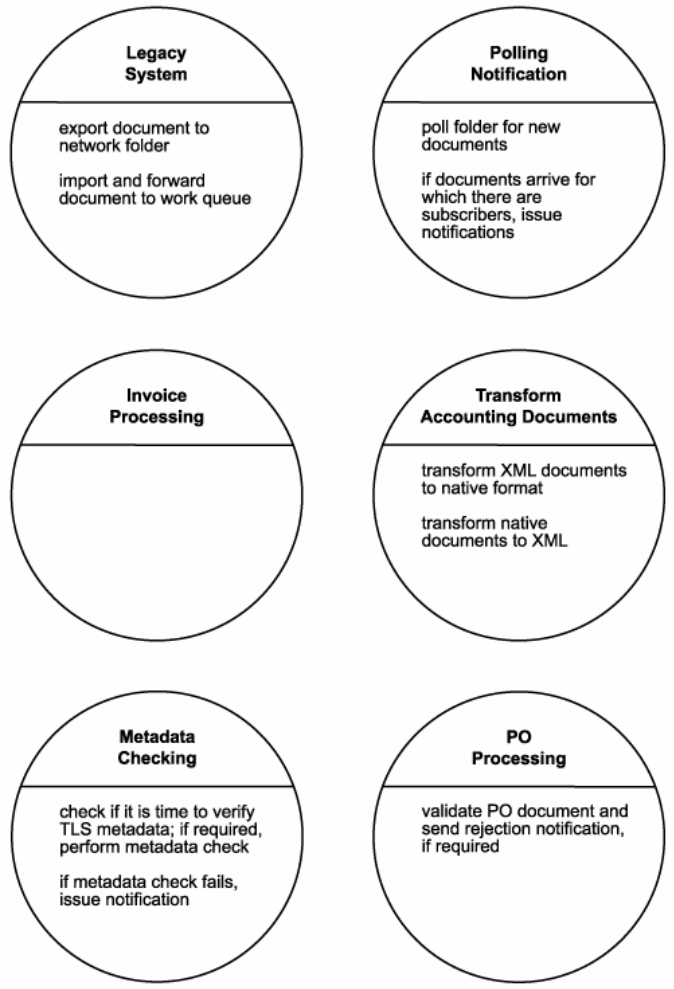
Step 5 RailCo example (12)

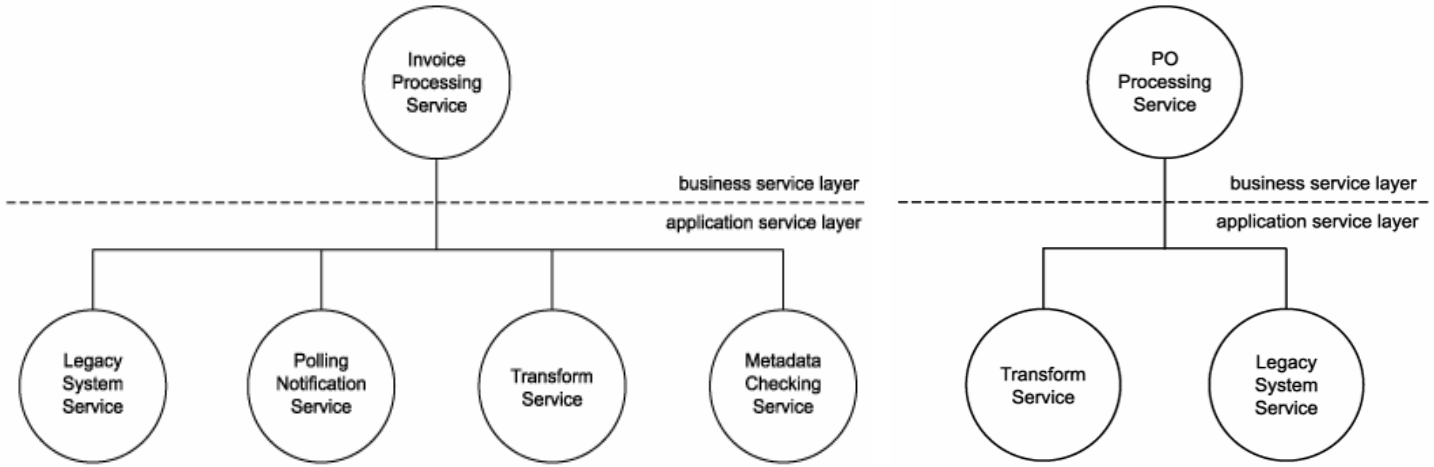
- Finally, our last group of operation candidates is reviewed. The candidates themselves are still relatively suitable for what we intended. However, because we've abstracted these into a generic service candidate, we need to revise the wording to better reflect this. Specifically, we add a notification feature to our Metadata Checking Service candidate.
- The revised Metadata Checking Service list contains the following steps:
  - Check if it is time to verify TLS metadata. If required, perform metadata check.
  - If metadata check fails, issue notification.

From service candidates to refined service candidates

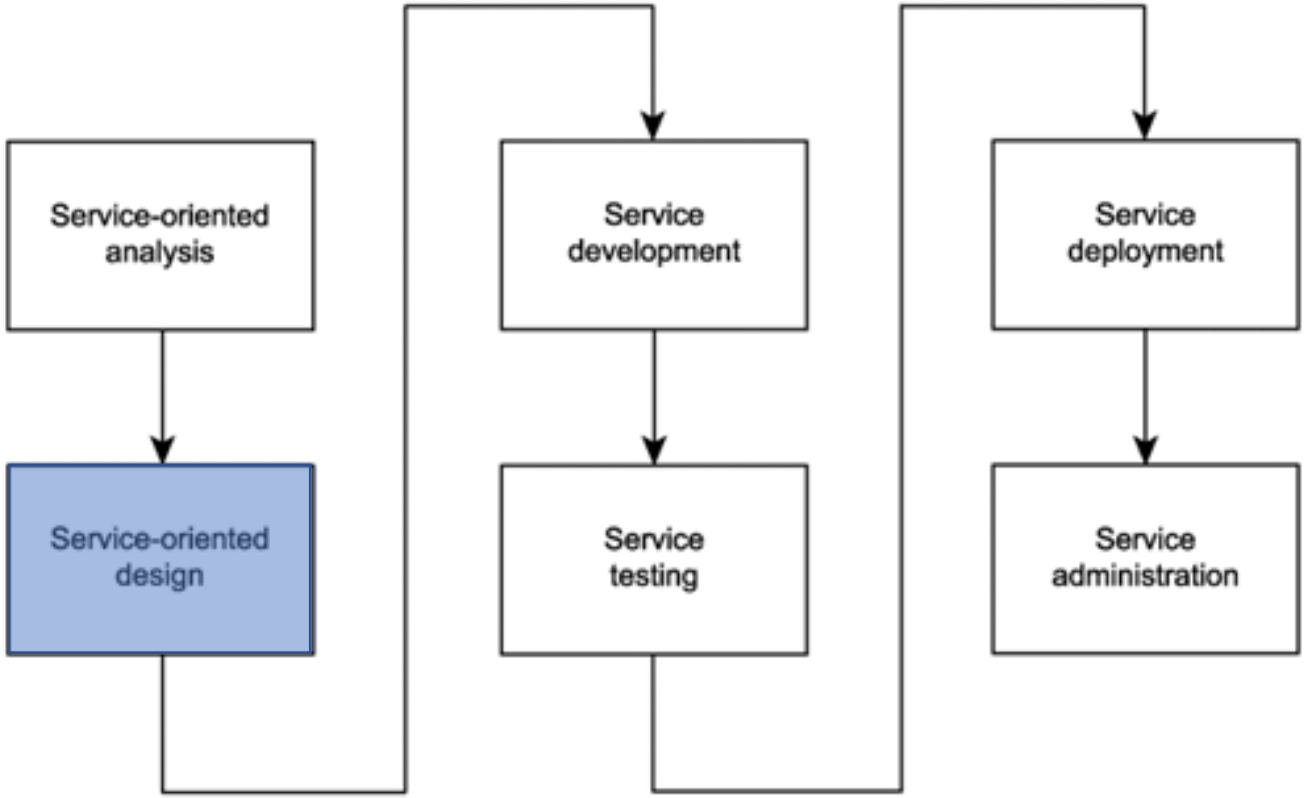


From service candidates to candidate service compositions





Service design

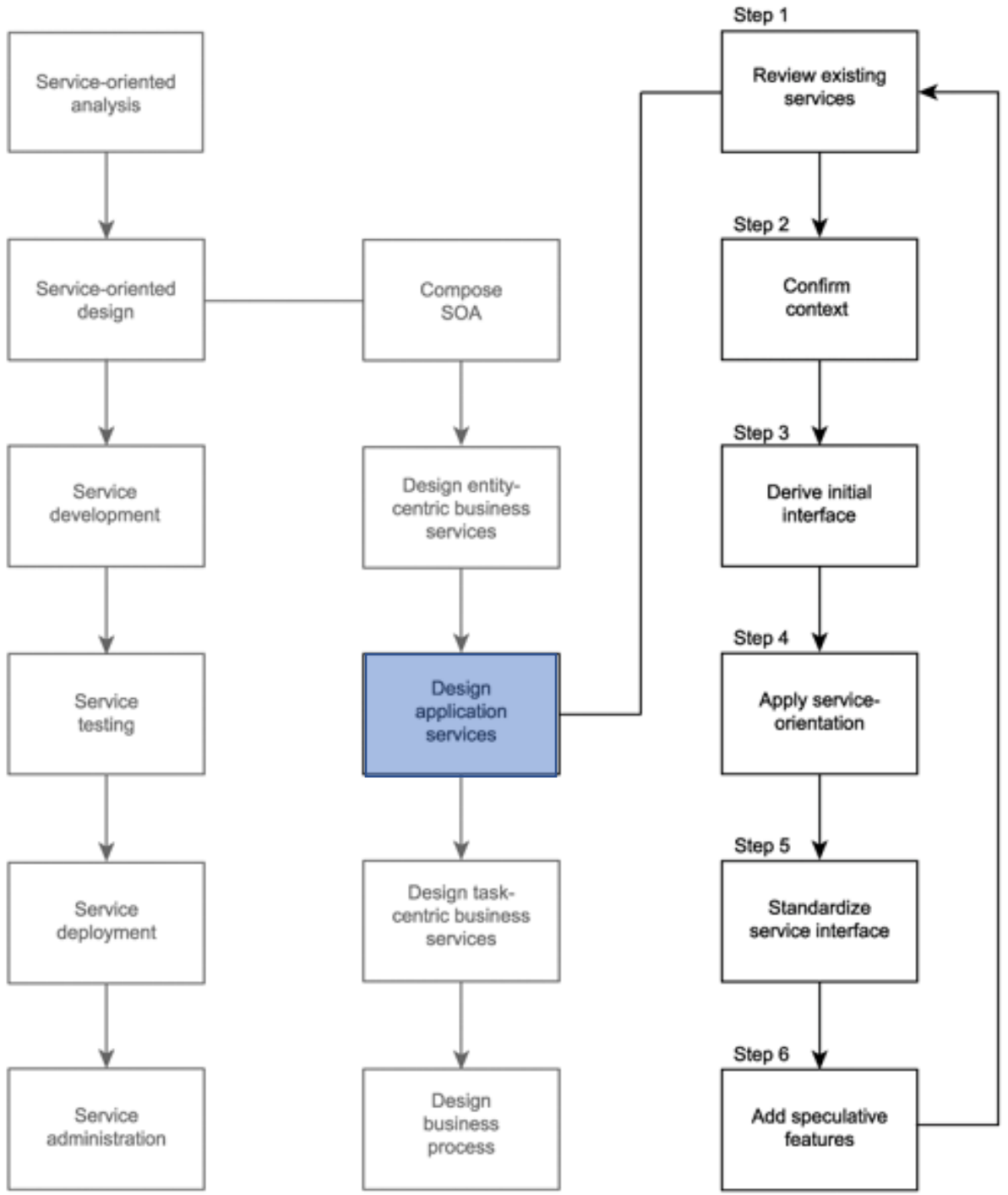


Service-oriented design

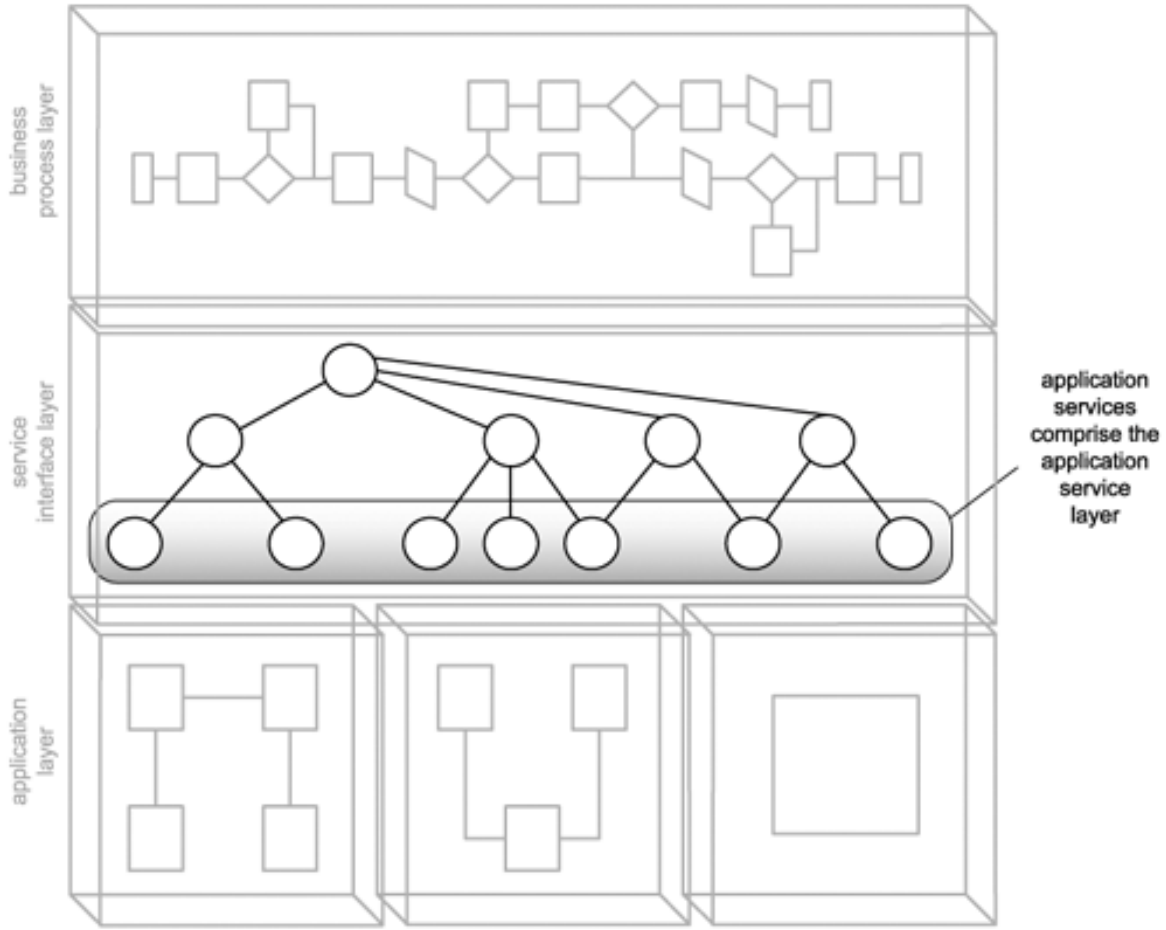
- Service-oriented design is the process by which concrete physical service designs are derived from logical service candidates and then assembled into abstract compositions that implement a business process.

Service-oriented design

Design application services

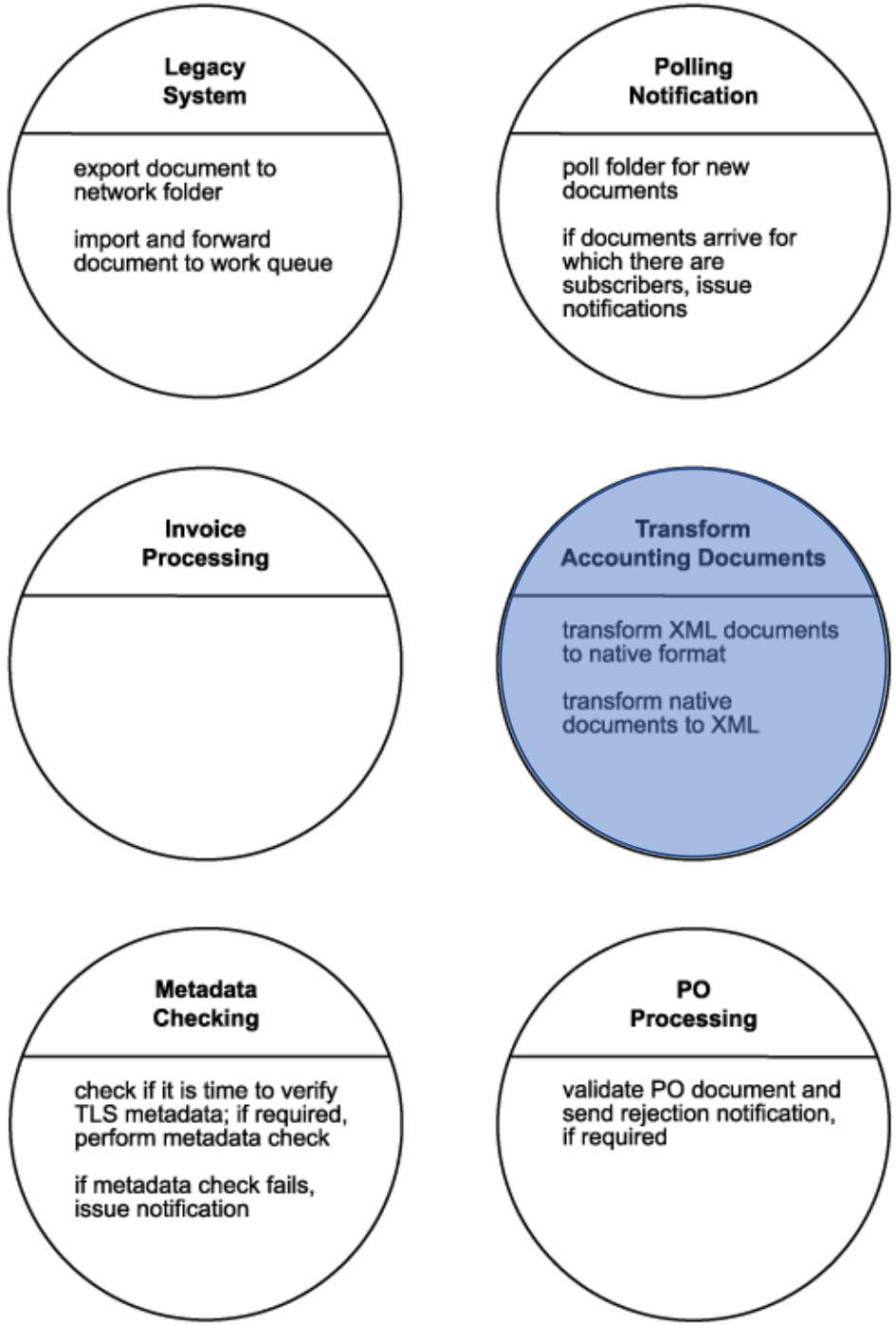


Design application services



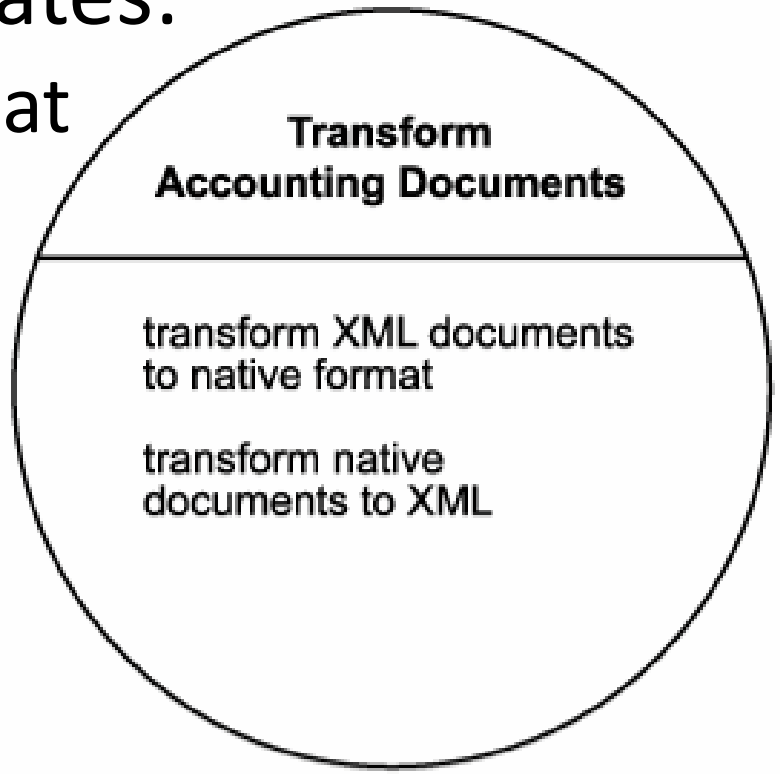
The design of the Transform application service candidate

- Are you able to perform the service design steps for the example scenario?

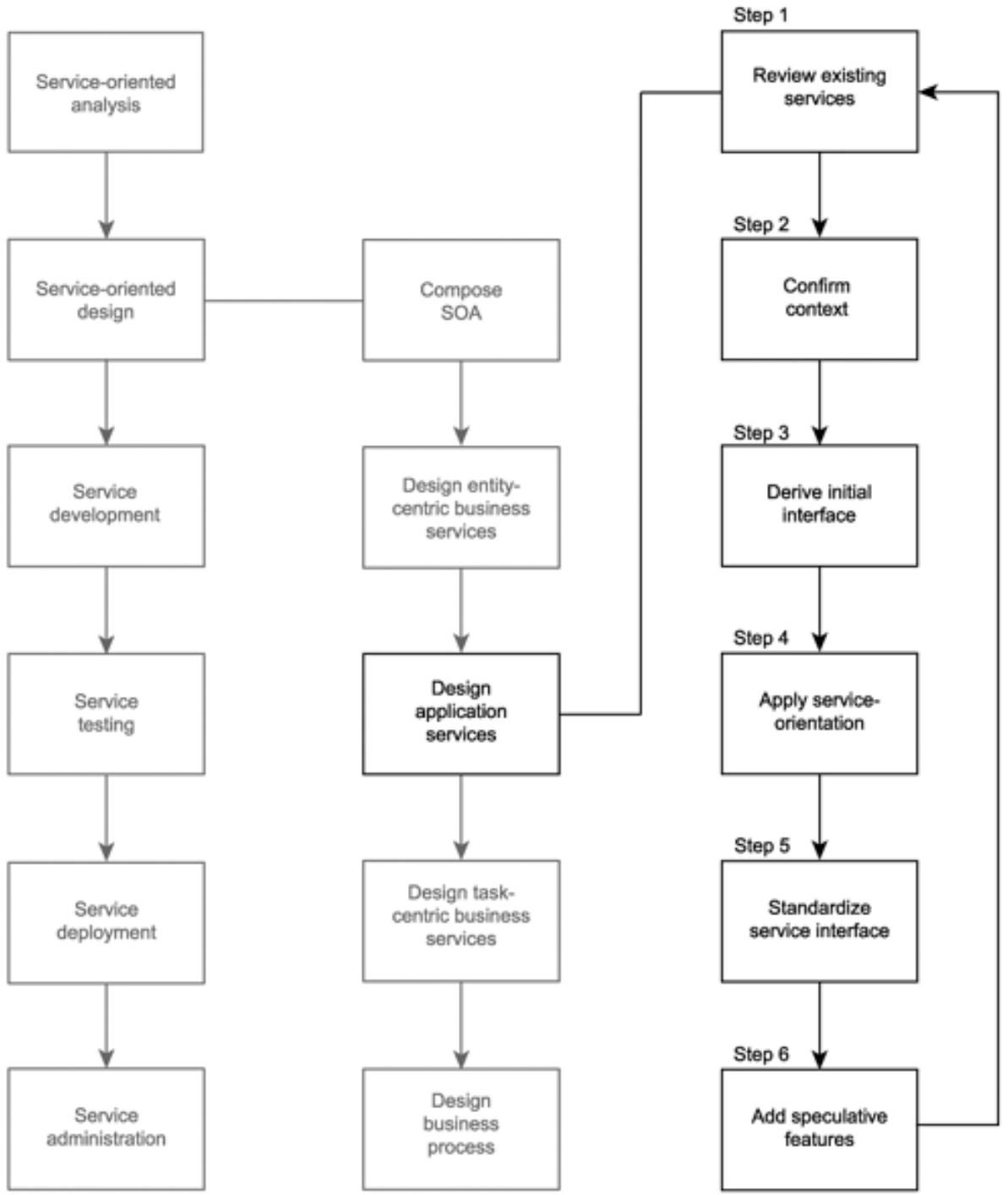

Legacy System
export document to network folder
import and forward document to work queue
Polling Notification
poll folder for new documents
if documents arrive for which there are subscribers, issue notifications
Invoice Processing
Transform Accounting Documents
transform XML documents to native format
transform native documents to XML
Metadata Checking
check if it is time to verify TLS metadata; if required, perform metadata check
if metadata check fails, issue notification
PO Processing
validate PO document and send rejection notification, if required

Service Design - Transform Service

- This candidate establishes a "document transformation context," which justifies the grouping of its two very similar operation candidates:
  - transform XML documents to native format
  - transform native documents to XML

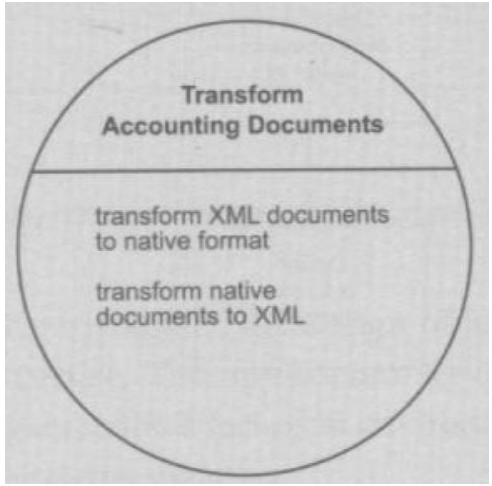


Service-oriented design



Step 1: Review existing services

- Check existing services for redundant features.
- Check if the required feature can be purchased or leased from vendors.



- Checking names, service descriptions and metadata of existing services (i.e. TLS Subscription Service), RailCo concludes that no overlap exists

Step 2: Confirm context

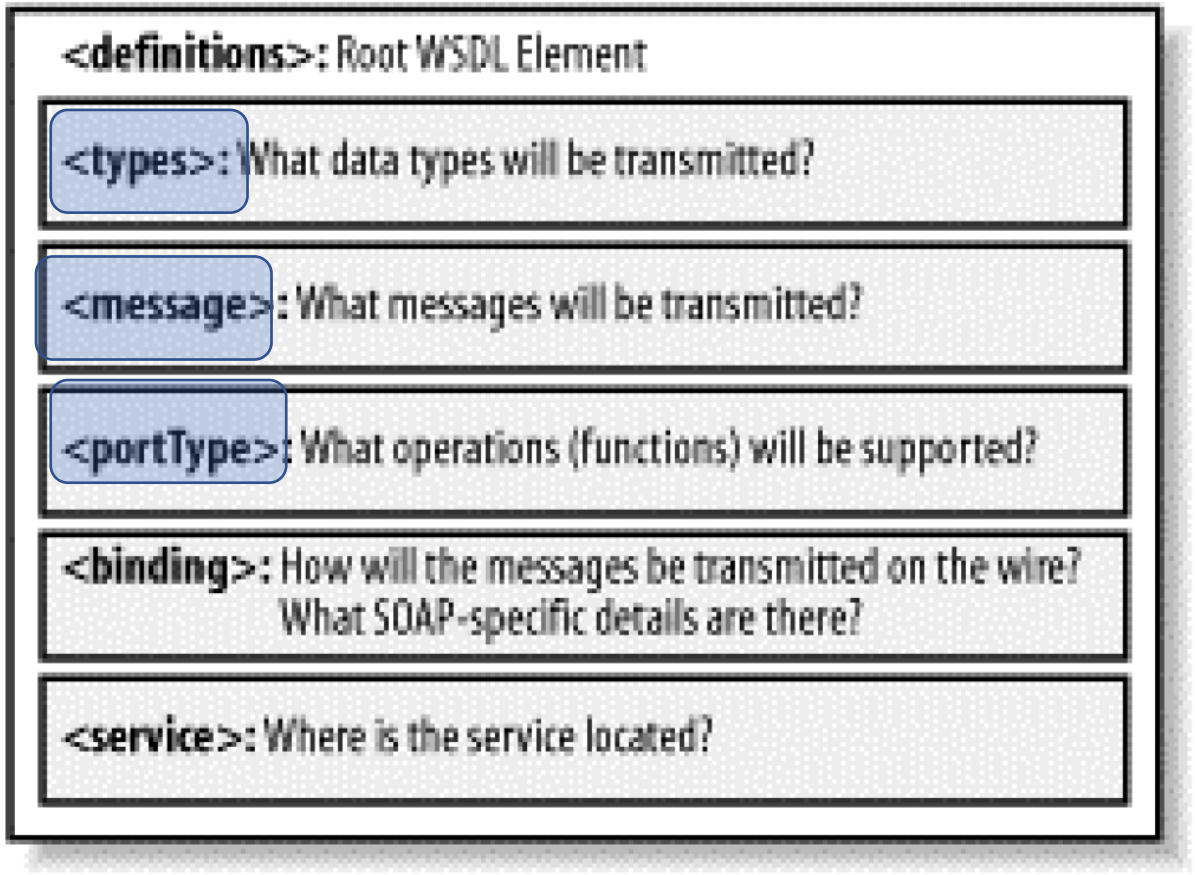
- Reassess service context if operation candidate grouping from the analysis phase is appropriate, e.g. operations may better belong in other services.
- Review of Transform Accounting Documents service against TLS Subscription service confirms that grouping context is valid.

Step 3: Derive an initial service interface

- Confirm that each operation candidate is suitably reusable and its granularity is appropriate.
  1. Define definition of messages the service is to process (i.e. WSDL <types> area) using XML schema.
  2. Establish a set of operation names and define WSDL <portType> with <operation>.
  3. Define WSDL <message> with <part>.

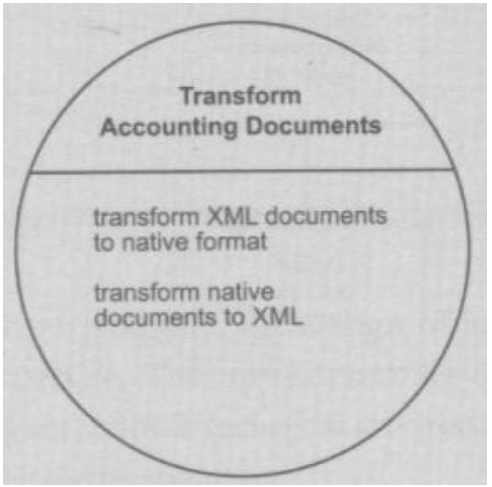
Step 3: Derive an initial service interface

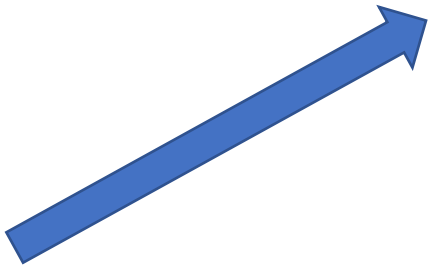
- Confirm that each operation candidate is suitably reusable and its granularity is appropriate.
  - Define definition of messages the service is to process (i.e. WSDL <types> area) using XML schema.
  - Establish a set of operation names and define WSDL <portType> with <operation>.
  - Define WSDL <message> with <part>.

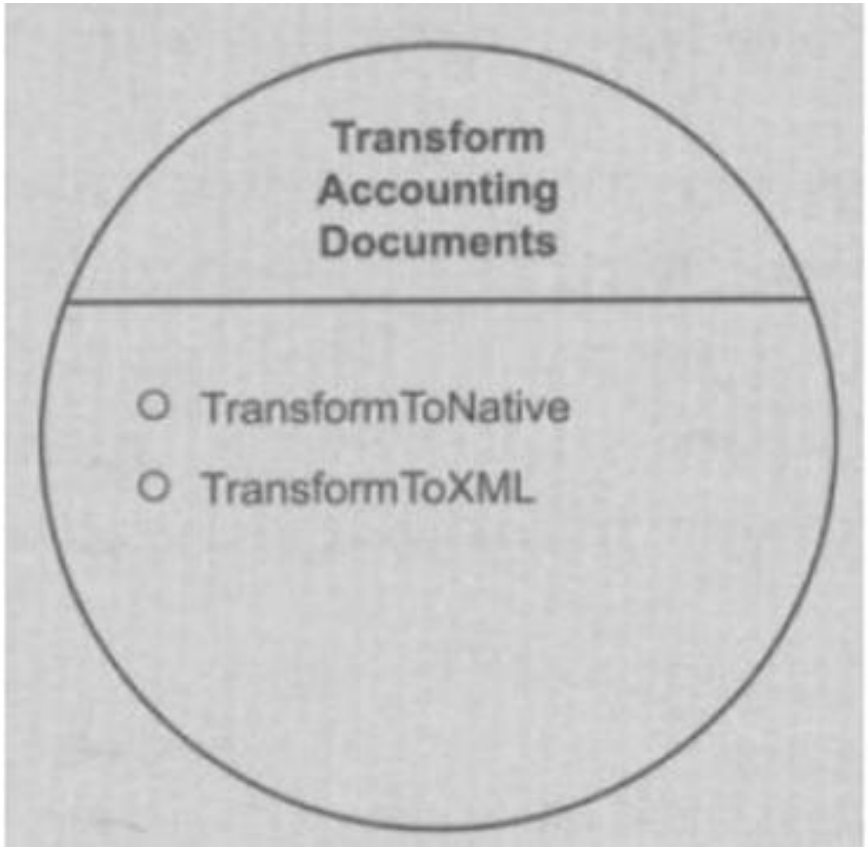


Confirm that each operation candidate is suitably reusable and its granularity is appropriate

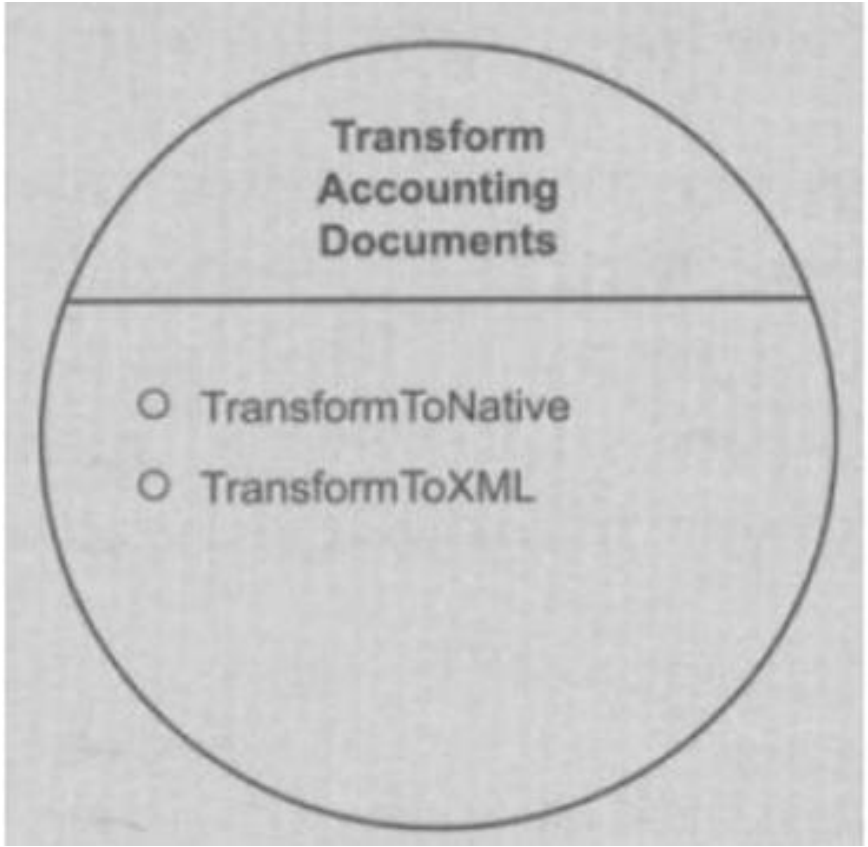
- RailCo begins by the two operation names shown here
- It then moves on to define the *types* construct of its service definition to formalize the message structures







Define definition of messages the service is to process (i.e. WSDL <types> area) using XML schema



<definitions>: Root WSDL Element

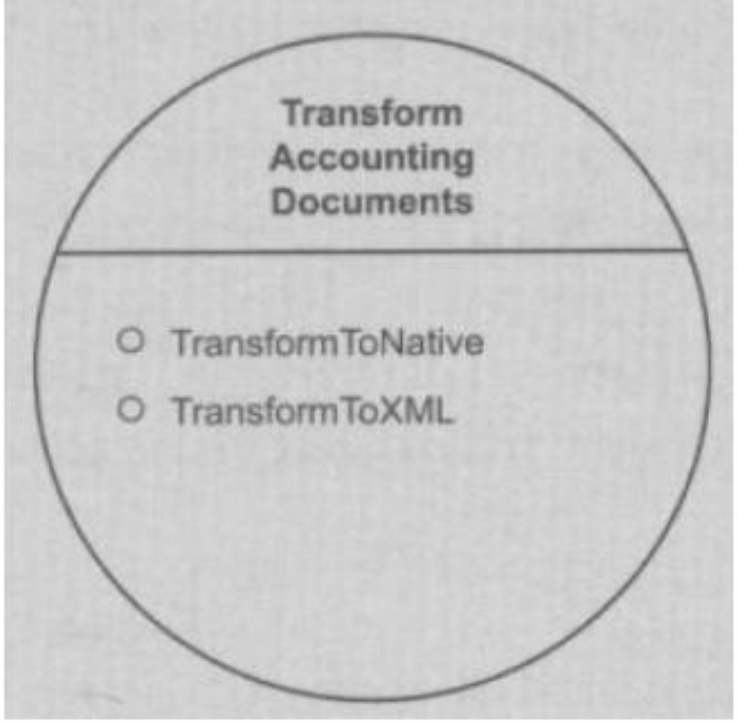
<types>: What data types will be transmitted?

<message>: What messages will be transmitted?

<portType>: What operations (functions) will be supported?

<binding>: How will the messages be transmitted on the wire? What SOAP-specific details are there?

<service>: Where is the service located?



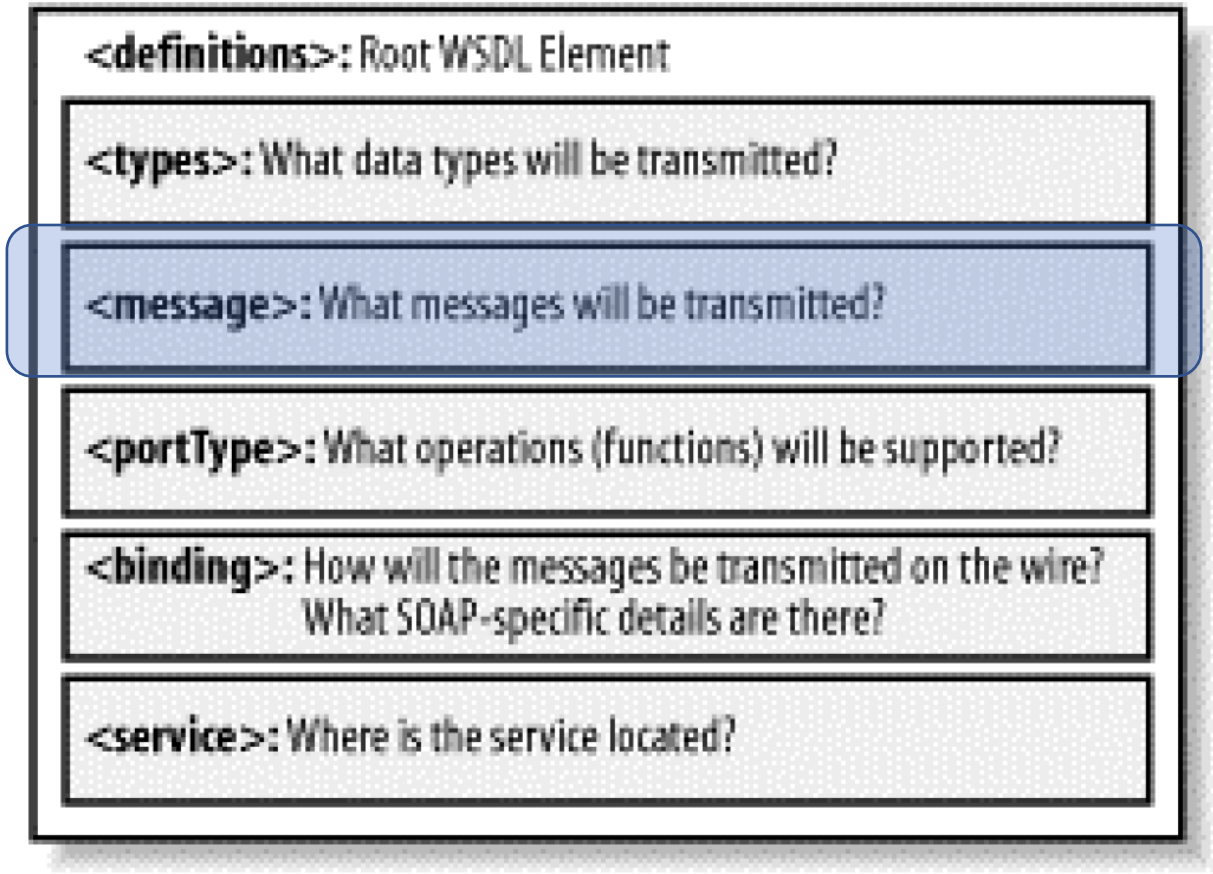
The request and response messages for the TransformtoNative operation

```
<xsd:schema targetNamespace=
   "http://www.xmltc.com/railco/transform/schema/">
   <xsd:element name="TransformToNativeType">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="SourcePath"
               type="xsd:string"/>
            <xsd:element name="DestinationPath"
               type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="TransformToNativeReturnCodeType">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Code"
               type="xsd:integer"/>
            <xsd:element name="Message"
               type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Types for TransformToXML are identical to those of TransformToNative and can be shared. But RailCo decides to keep a separate schema to have freedom to change them independently.

```
<xsd:element name="TransformToXMLType">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="SourcePath"
            type="xsd:string"/>
         <xsd:element name="DestinationPath"
            type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
<xsd:element name="TransformToXMLReturnCodeType">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="Code"
            type="xsd:integer"/>
         <xsd:element name="Message"
            type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

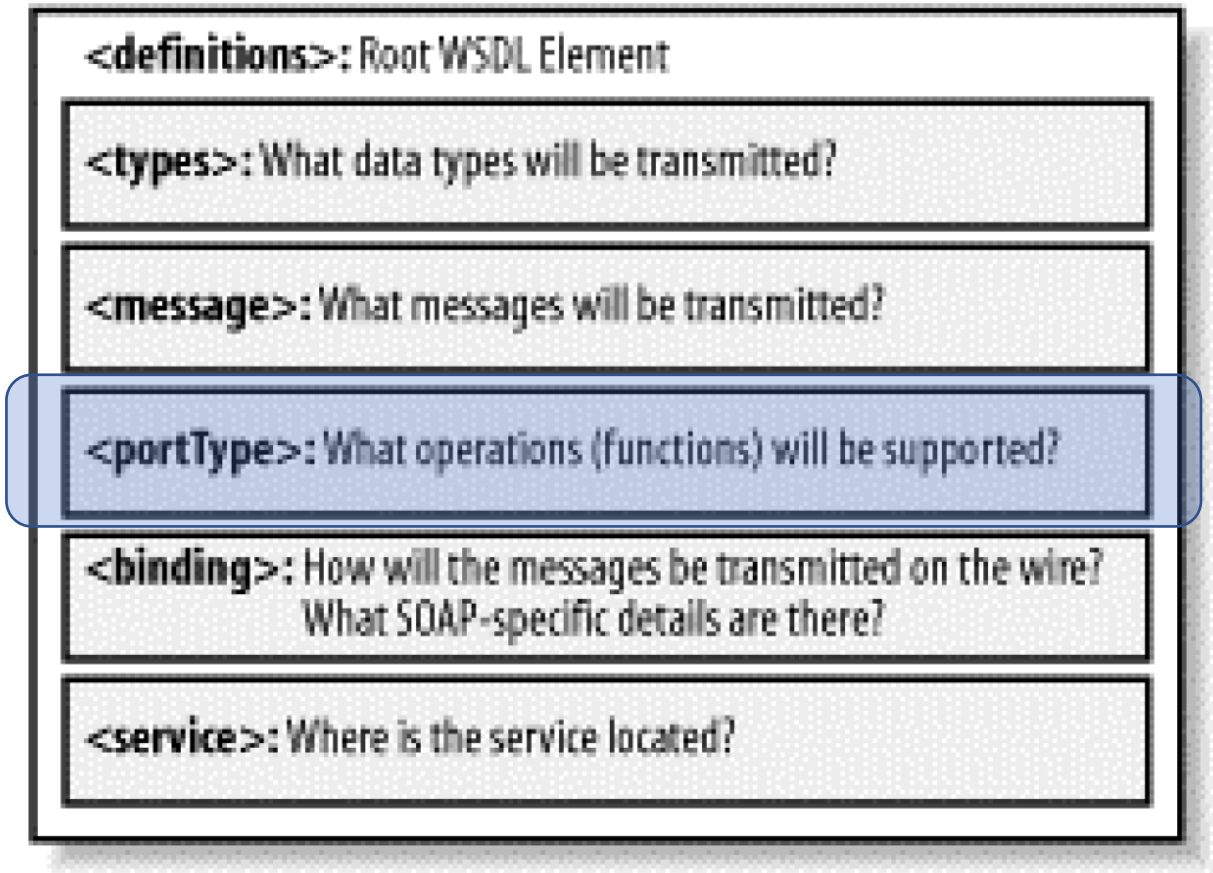
Define WSDL <message> with <part>




```
<message name="transformToNativeRequestMessage">
   <part name="RequestParameter"
      element="trn:TransformToNativeType"/>
</message>
<message name="transformToNativeResponseMessage">
   <part name="ResponseParameter"
      element="trn:TransformToNativeReturnCodeType"/>
</message>

<message name="transformToXMLRequestMessage">
   <part name="RequestParameter"
      element="trn:TransformToXMLType"/>
</message>
<message name="transformToXMLResponseMessage">
   <part name="ResponseParameter"
      element="trn:TransformToXMLReturnCodeType"/>
</message>
```

Establish a set of operation names and define WSDL <portType> with <operation>




```
<portType name="TransformInterface">
   <operation name="TransformToNative">
      <input message=
            "tns:transformToNativeRequestMessage"/>
         <output message=
            "tns:transformToNativeResponseMessage"/>
   </operation>
   <operation name="TransformToXML">
      <input message=
         "tns:transformToXMLRequestMessage"/>
      <output message=
          "tns:transformToXMLResponseMessage"/>
    </operation>
</portType>
```

Step 4: Apply service-orientation

- Revisit reusability, autonomy, statelessness, discoverability
- Reuse potential - the two operations TransformToNative and TransformToXML are discussed to be combined into one generic operation. But RailCo decides to keep them both for the descriptive nature of the operations and freedom to evolve each of them separately
- Autonomy and statelessness - no problem as the logic required to carry out the transformation is contained within the particular underlying application logic
- Discoverability - metadata are added

Service definition is outfitted with additional metadata documentation

```
<portType name="TransformInterface">
   <documentation>
      Retrieves an XML document and converts it
      into the native accounting document format.
   </documentation>
   <operation name="TransformToNative">
      <input message=
         "tns:transformToNativeRequestMessage"/>
      <output message=
         "tns:transformToNativeResponseMessage"/>
   </operation>
   <documentation>
      Retrieves a native accounting document and
      converts it into an XML document.
   </documentation>
   <operation name="TransformToXML">
      <input message=
         "tns:transformToXMLRequestMessage"/>
      <output message=
         "tns:transformToXMLResponseMessage"/>
   </operation>
</portType>
```

Step 5: Standardize service interface

- Apply appropriately any existing design standards and guidelines.
  - E.g. naming convention: Names should be generic for high reuse potential and clearly communicate the processing context (verb+noun or noun).
- Consider quality service design, e.g. extensibility.

Step 5: Standardize service interface

- RailCo thinks service and operation names are already appropriate. However, the names are changed for future extensibility. The service becomes a transformation utility that can be extended in the future to offer other transformation related features (not only for use by an accounting system).

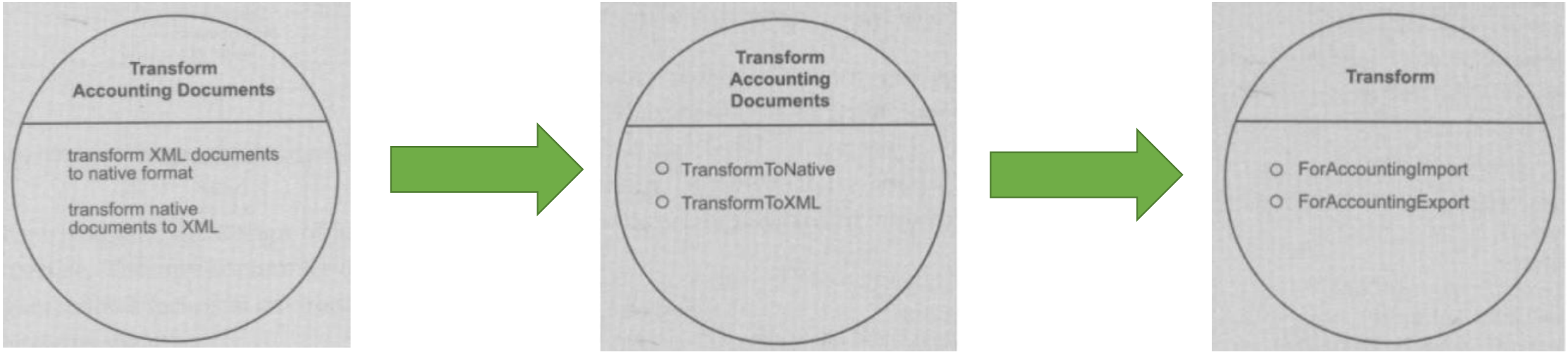
```
<types>
   <xsd:schema targetNamespace=
      "http://www.xmltc.com/railco/transform/schema/">
      <xsd:element name="ForImportType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="SourcePath"
                  type="xsd:string"/>
               <xsd:element name="DestinationPath"
                  type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="ForImportReturnCodeType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="Code"
                  type="xsd:integer"/>
               <xsd:element name="Message"
                  type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
```

```
        <xsd:element name="ForExportType">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="SourcePath"
                        type="xsd:string"/>
                    <xsd:element name="DestinationPath"
                        type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="ForExportReturnCodeType">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="Code"
                        type="xsd:integer"/>
                    <xsd:element name="Message"
                        type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:schema>
</types>
```

Step 6: Add speculative features

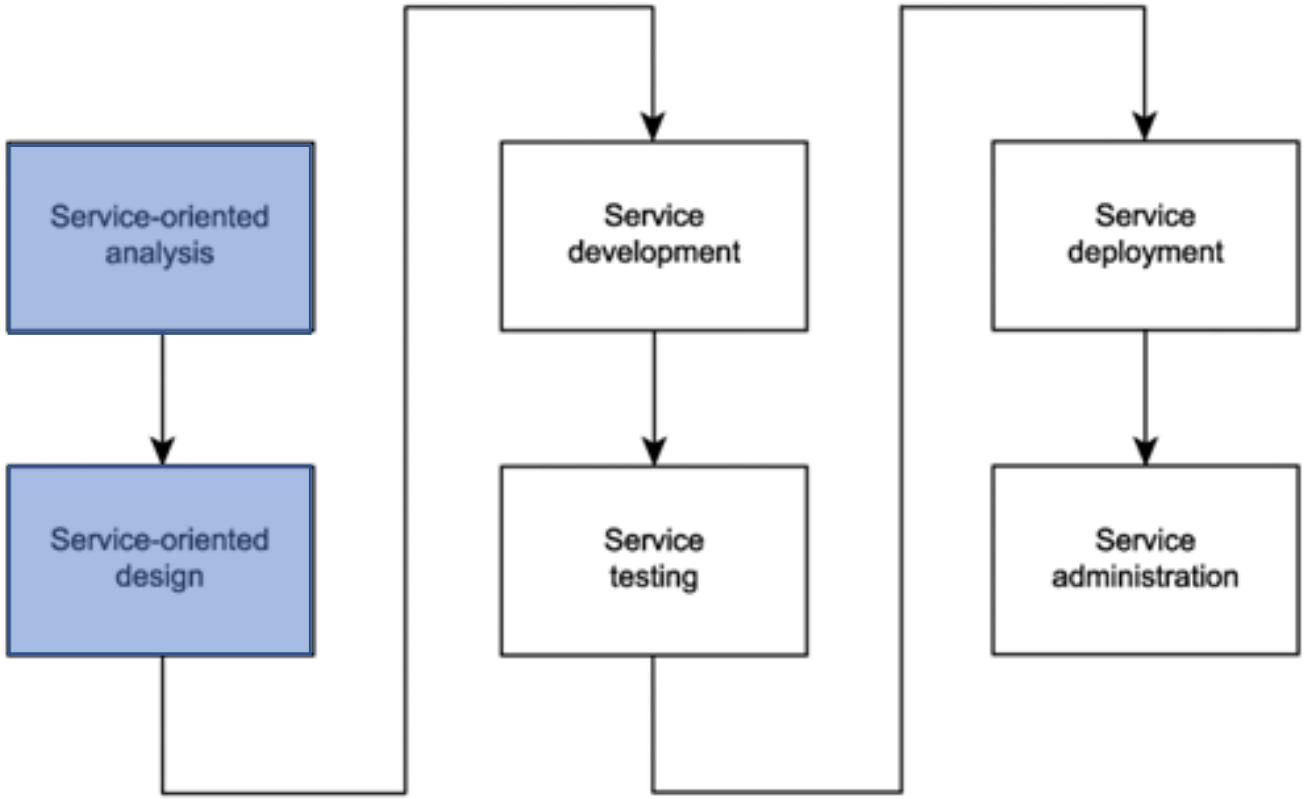
- Perform speculative analysis as to what other processing falls within the service context.
- Repeat steps 1-5.
- RailCo cannot afford any extension at this time. But the design of Transform service already gives future reusability opportunity.

Further refinements during service design



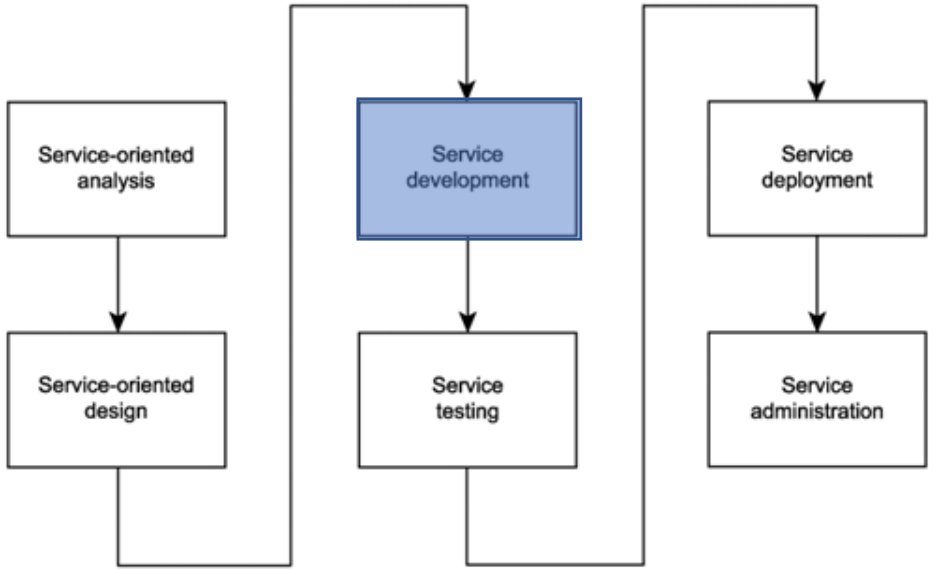
Why some tasks are repeated during both phases: analysis and design?

- Who should participate in these step?
- What kind of expertise is required?



Service development

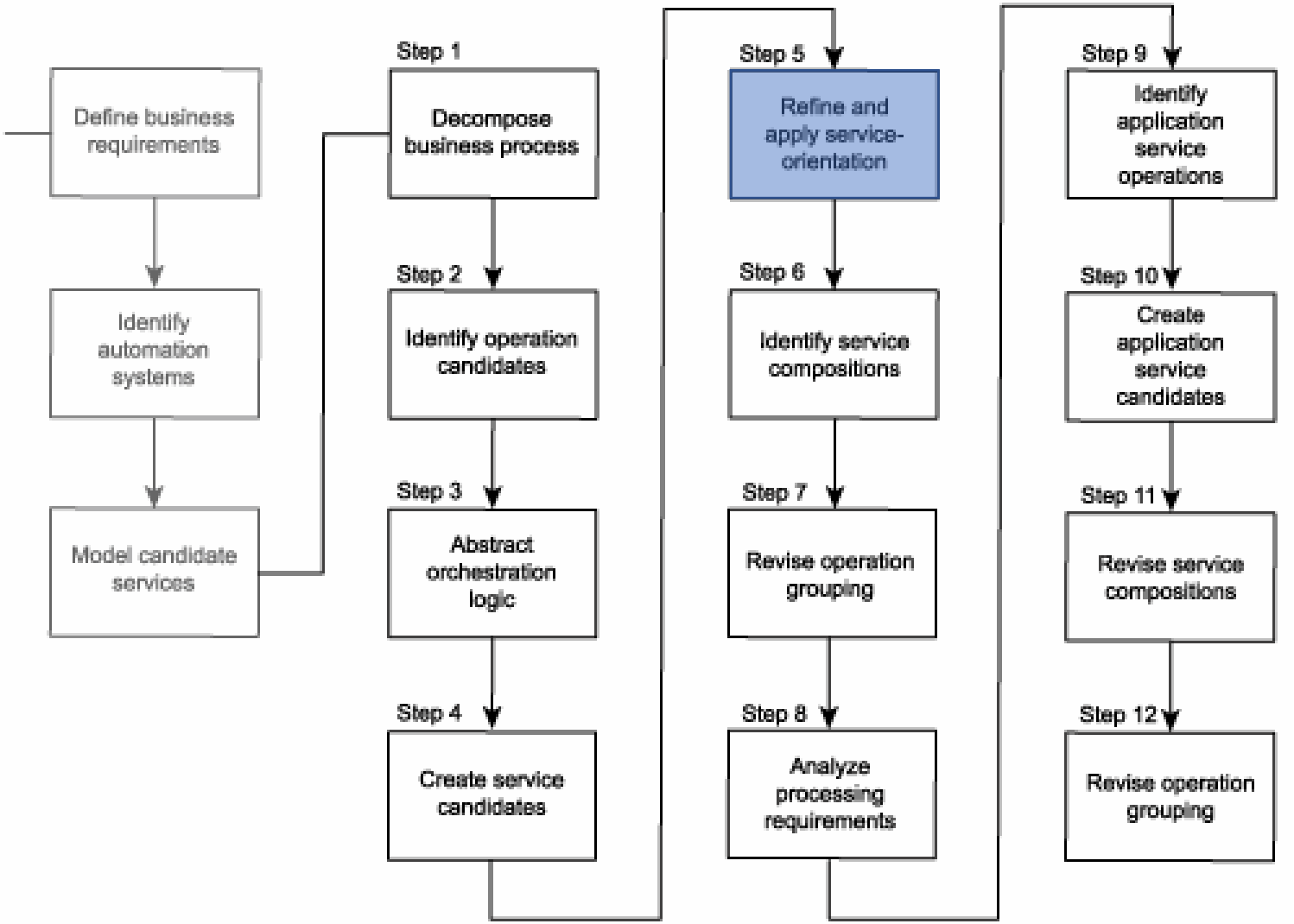
- What development technologies do you know?
  - Check Module 4 ppt slides to review this issue



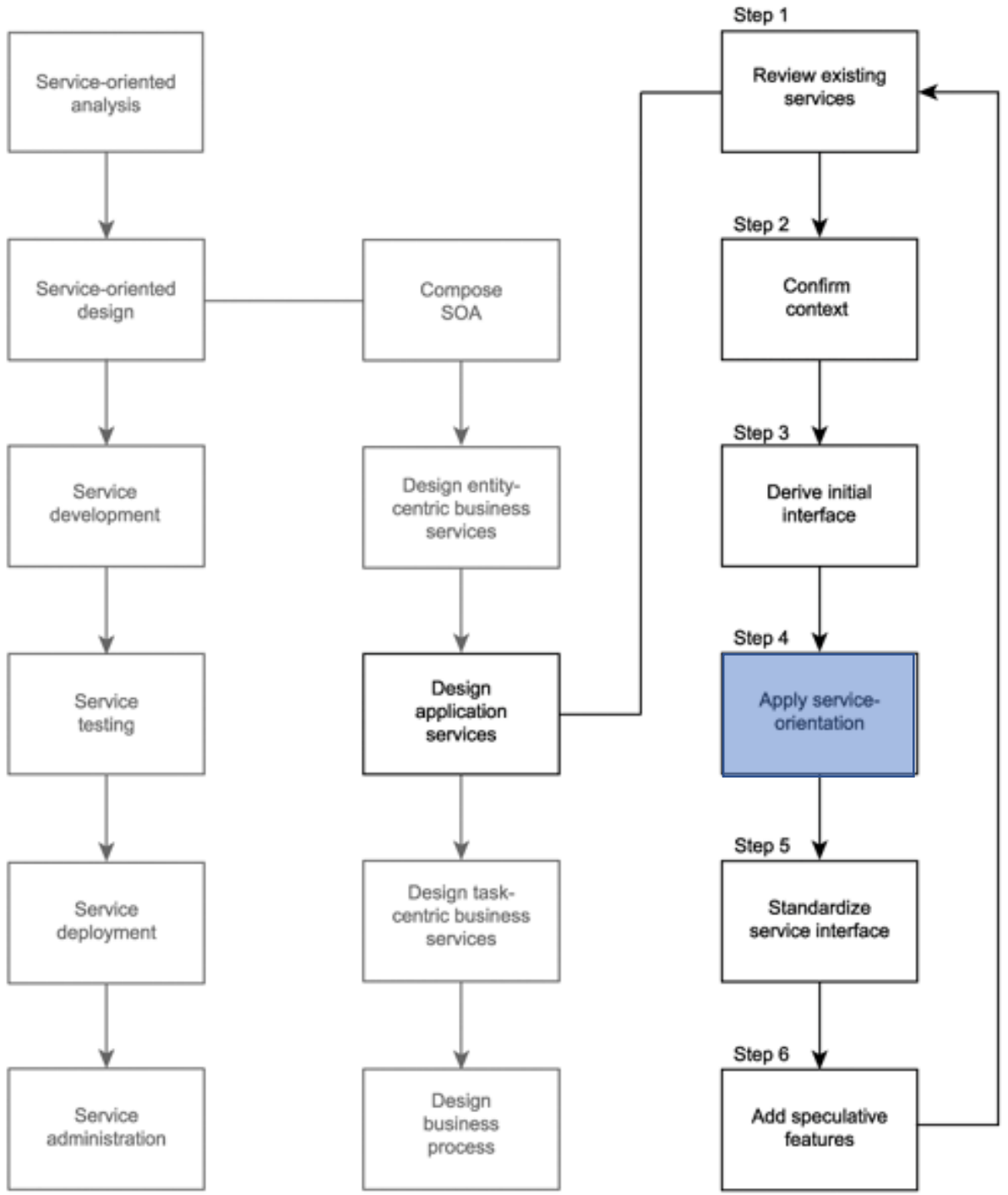
Discussion

- Service orientation was already applied during the analysis phase. Why do we need to do it again?

Service oriented analysis

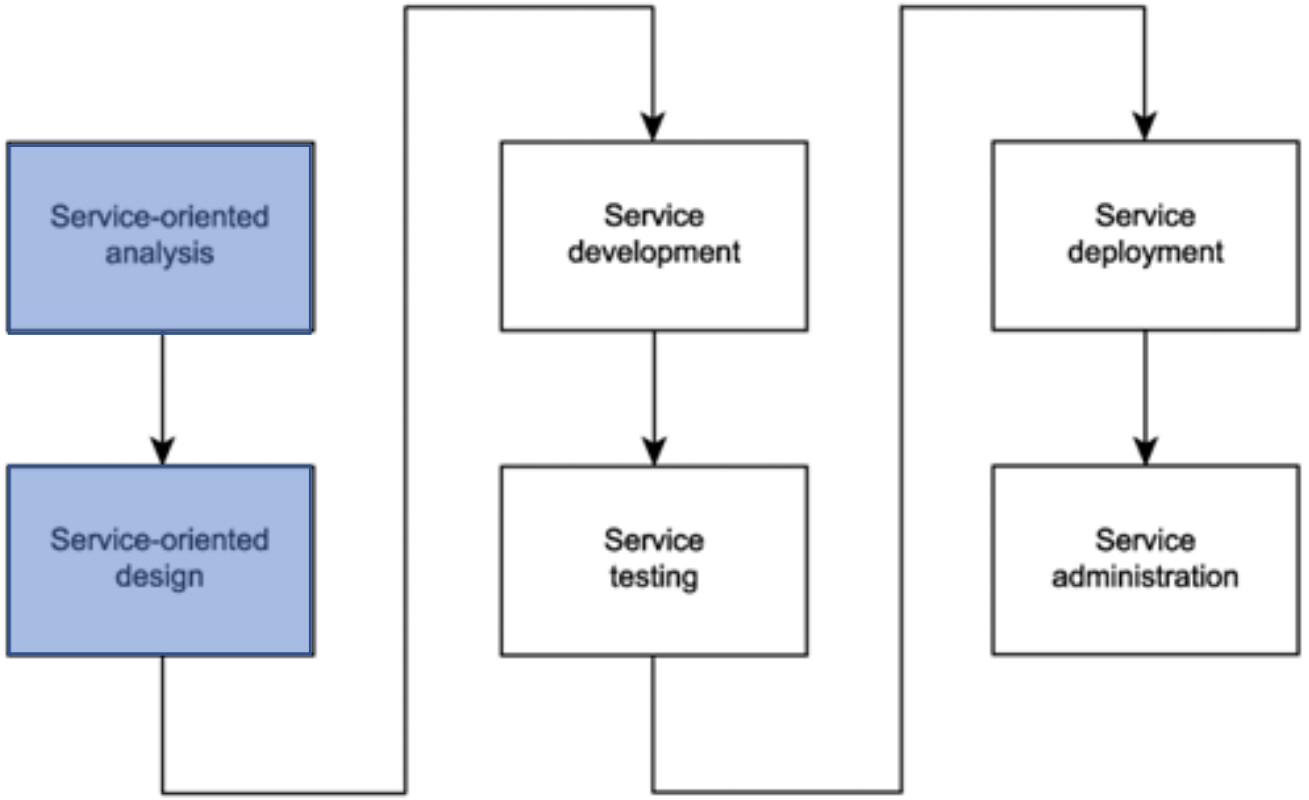


Service-oriented design



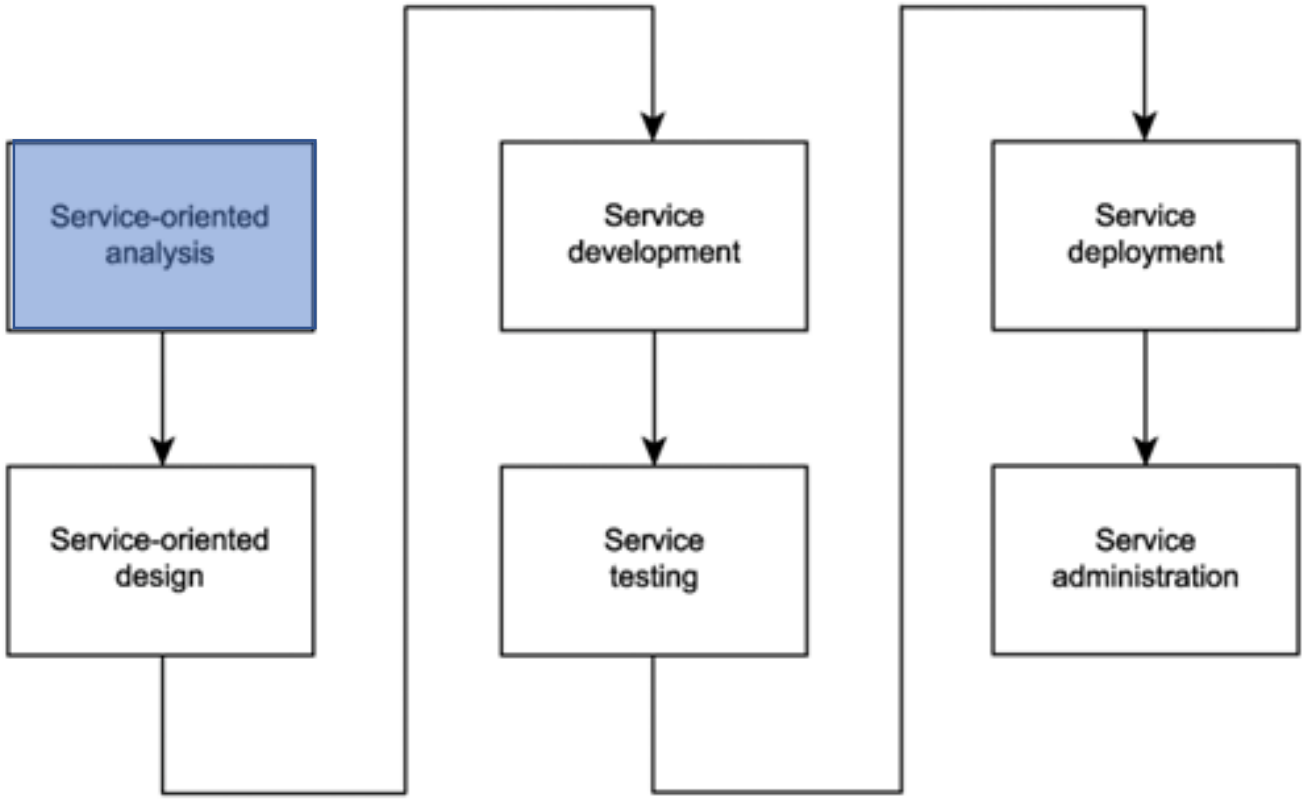
Service orientation was already applied during the analysis phase. Why do we need to do it again?

- We need to consider the following questions first
  - Who should participate in these steps?
  - What kind of expertise is required?



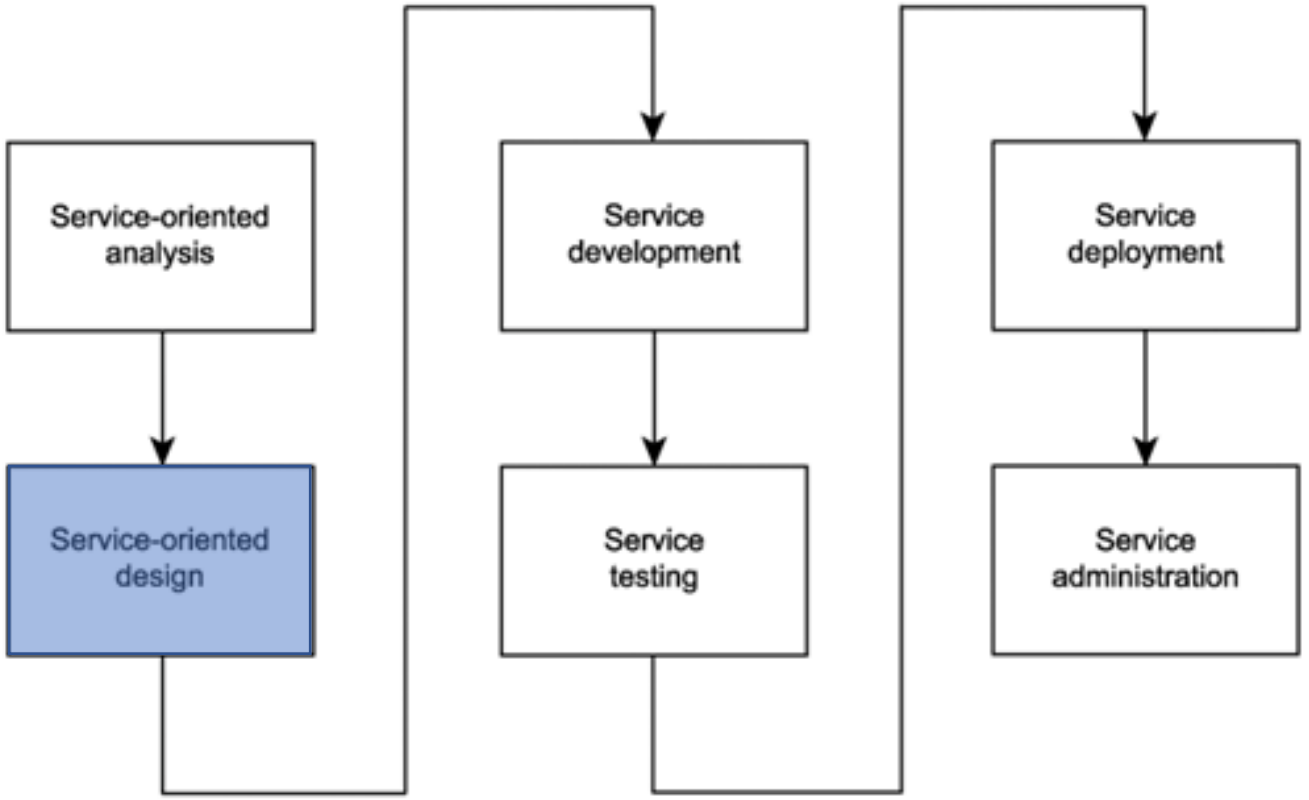
Who should participate in service analysis?

- Business analysis expertise is required



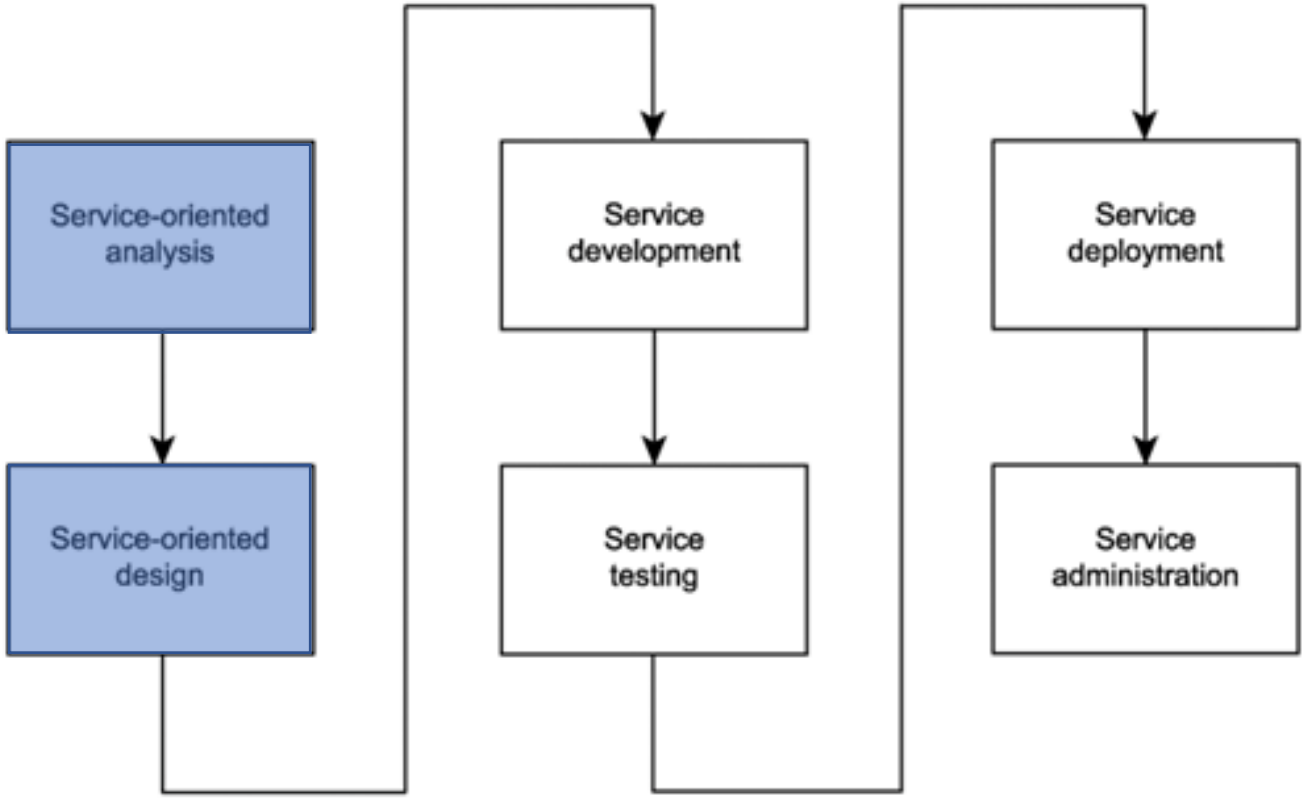
Who should participate in service design?

- Best defined by those who understand the enterprise technical environments the most
- Business analysis expertise not required

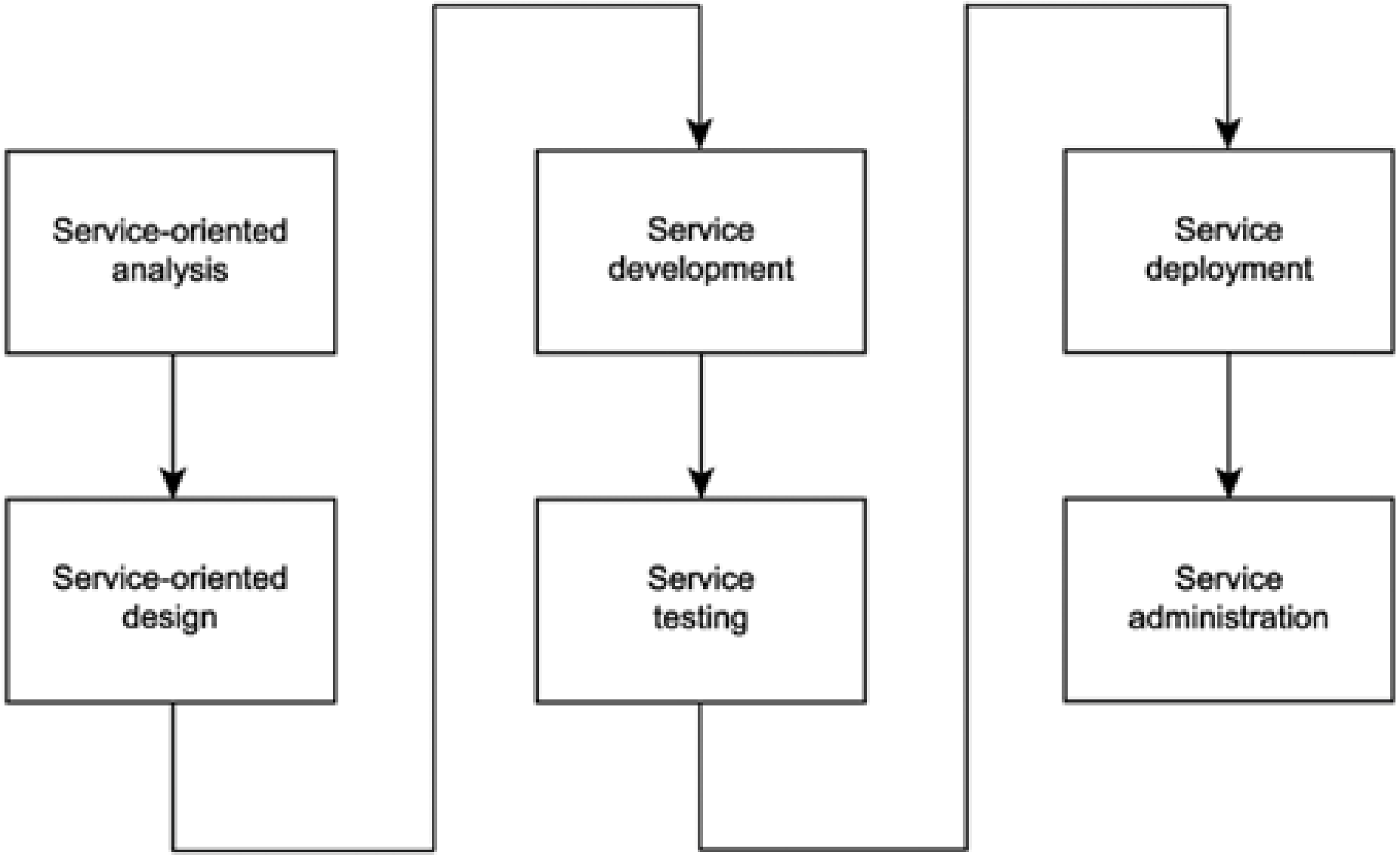


What expertise is required?

- Business
- Technical



Summary


Service-oriented analysis
Service-oriented design
Service development
Service testing
Service deployment
Service administration