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
  - All members should contribute to creating the homepage
  - The members of the group can edit/ each other's work in the home page
  - You can use information from the overview you created in task 1 (journal), combine it and organize it with information from other group members
  - You can also find new information to add to your wiki homepage
  - You can add figures/links/multimedia
- Deadline: 28th April (Thursday)

Module Six: Service Engineering

Our materials for this module:

- **Thomas Erl, Service-Oriented Architecture, Concepts, Technology, and Design, ISBN： 9787030336422，**
  - Chapter 10
  - Chapter 11.1 and 11.2
  - Chapter 12.1
  - Chapter 13.1 and 13.5
  - Chapter 14.1
  - Chapter 15.1, 15.2, 15.3, and15.4
  - Chapter 16.3

Module 6 Learning Outcomes

- Understand the common phases of an SOA delivery lifecycle
- Understand the difference between different SOA delivery strategies
- Be able to conduct a service-oriented analysis
- Be able to use WSDL, SOAP and BPEL in service design

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 happens in service-oriented design phase of an SOA delivery lifecycle?
- What choices are made during service development?
- What key issue face service testers?
- What issues arise during service deployment?
- What application management issues come to the forefront during service administration?

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?
- How do we use WSDL, SOAP and BPEL in service design?

SOA Lifecycle

Service Engineering

- also called service-oriented software engineering
- a software engineering process that attempts to decompose the system into self-running units that either perform services or expose services

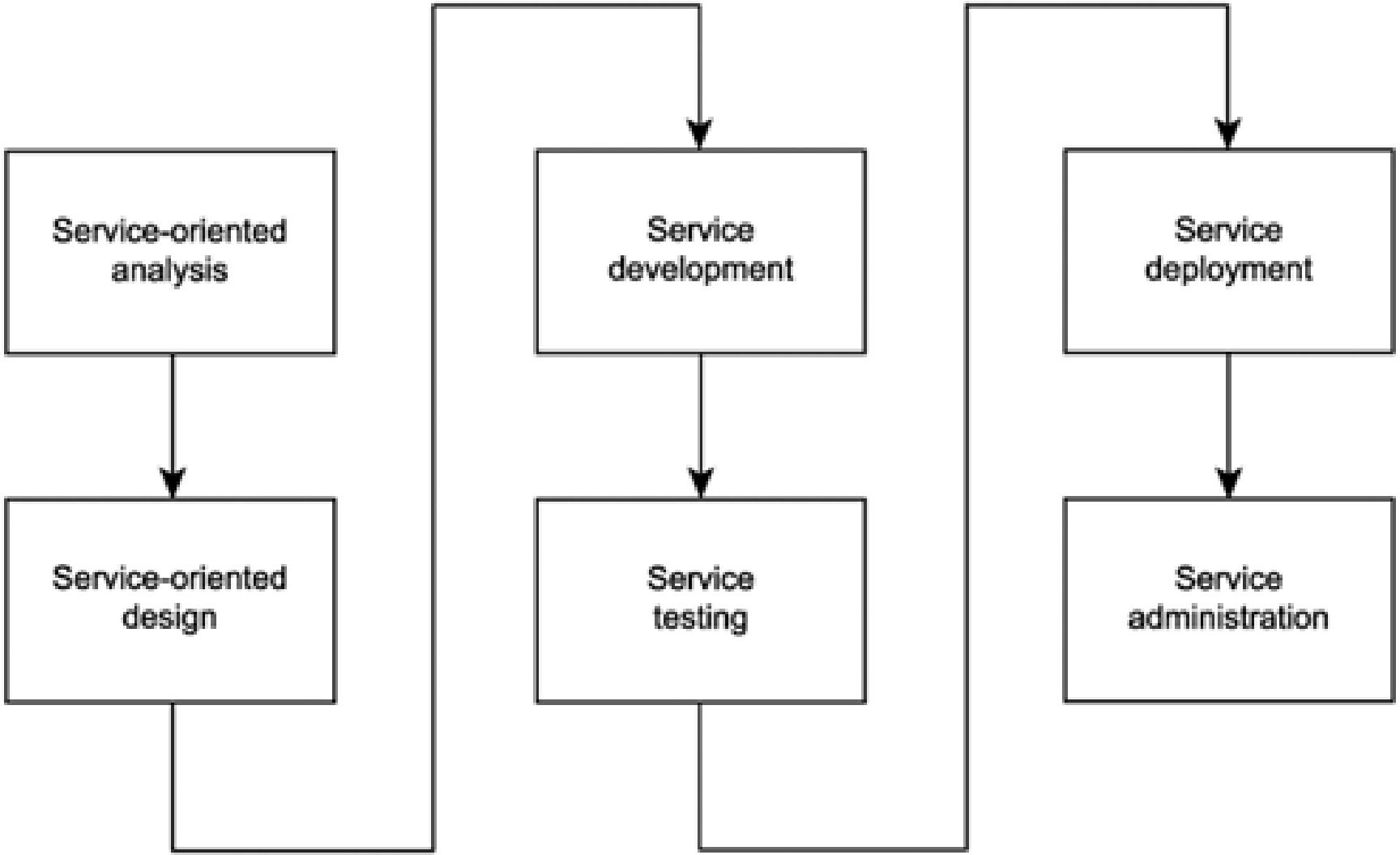
SOA lifecycle

- A series of steps that need to be completed to construct the services for a given service-oriented solution.
- The basic SOA lifecycle consists of a series of phases similar to those used for regular development projects

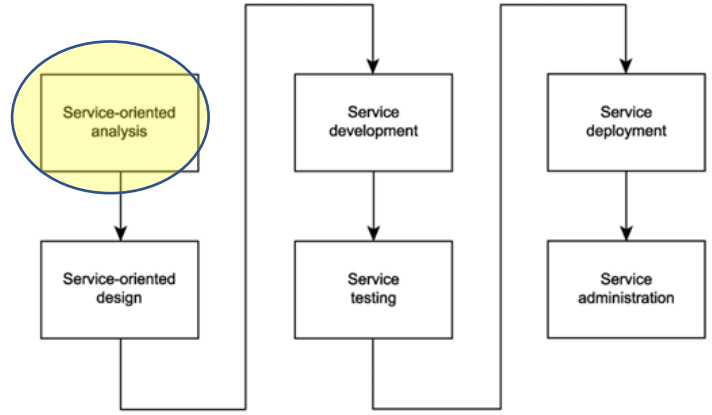
SOA lifecycle

- SOA introduces unique considerations in every phase of service construction and delivery

Common phases of an SOA delivery lifecycle

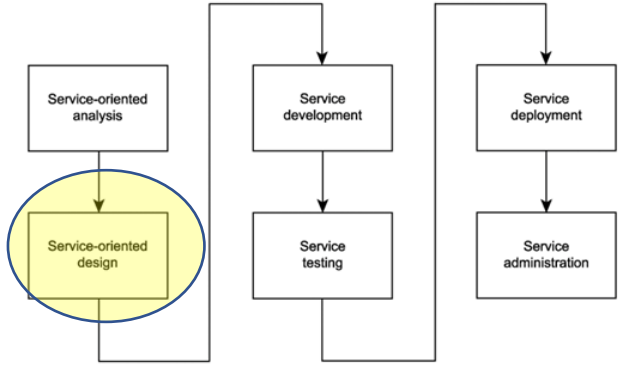


Service-oriented analysis



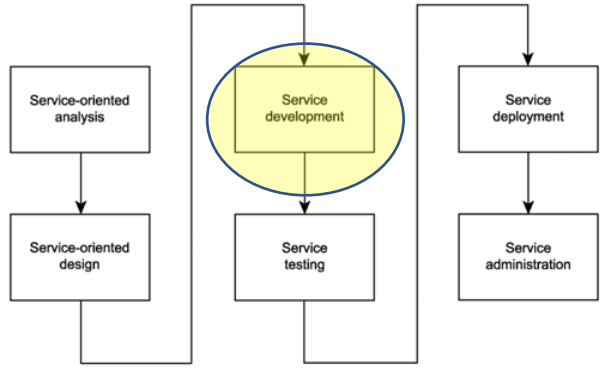
- In this initial stage that we determine the potential scope of our SOA
- Service layers are mapped out
- Individual services are modeled as service candidates that comprise a preliminary SOA

Service-oriented design



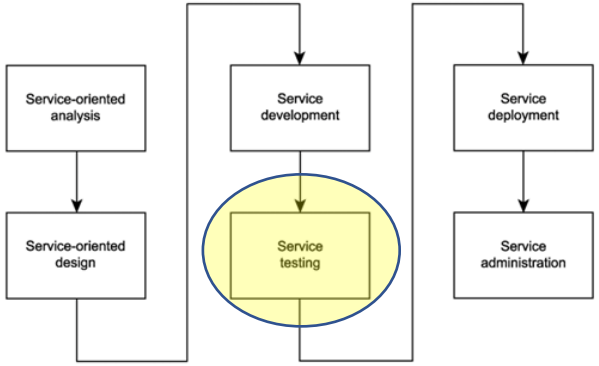
- When we know what it is we want to build, we need to determine how it should be constructed.
- Service-oriented design is a heavily standards-driven phase that incorporates industry conventions and service-orientation principles into the service design process.
- This phase, therefore, confronts service designers with key decisions that establish the hard logic boundaries encapsulated by services. The service layers designed during this stage can include the orchestration layer, which results in a formal business process definition.

Service development



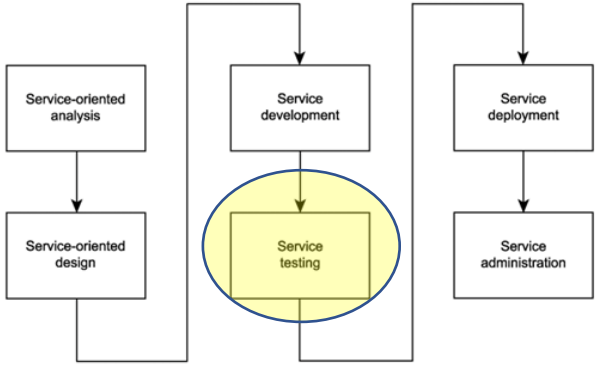
- The actual construction phase.
- Development platform-specific issues come into play, regardless of service type.
  - The choice of programming language and development environment will determine the physical form services and orchestrated business processes take, in accordance with their designs.

Service testing (1)



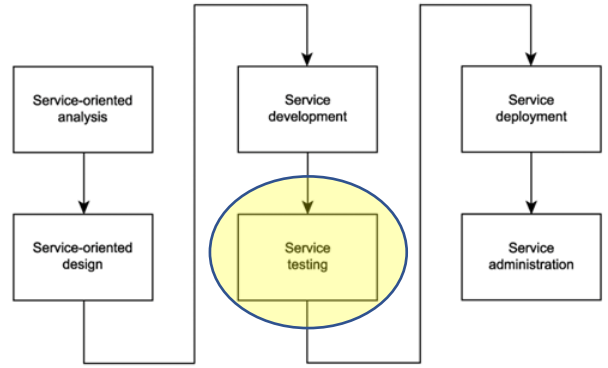
- Given their generic nature and potential to be reused and composed in unforeseeable situations, services are required to undergo rigorous testing prior to deployment into a production environment.

Service testing (2)



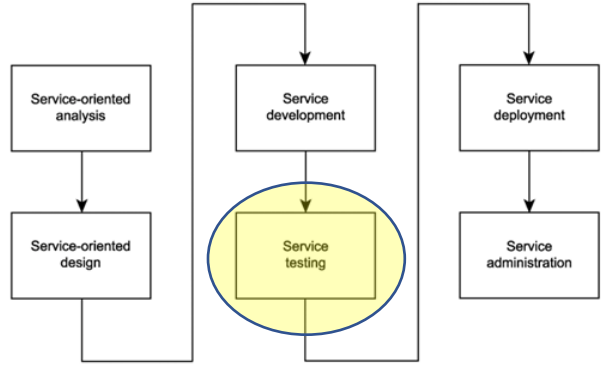
- Some of the key issues facing service testers:
  - What types of service requestors could potentially access a service?
  - Can all service policy assertions be successfully met?
  - What types of exception conditions could a service be potentially subjected to?
  - How well do service descriptions communicate service semantics?
  - Do revised service descriptions alter or extend previous versions?

Service testing (3)



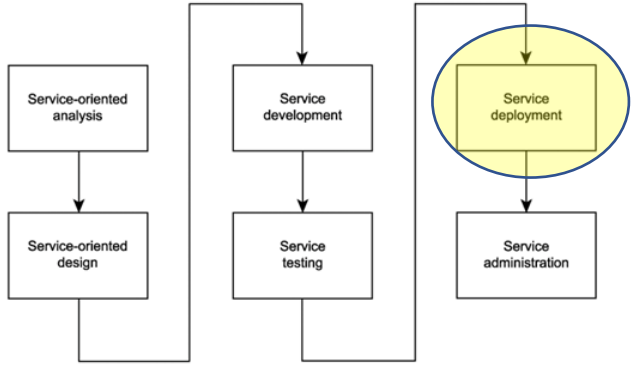
- Some of the key issues facing service testers:
  - How easily can the services be composed?
  - How easily can the service descriptions be discovered?
  - Is compliance to WS-I profiles required?
  - What data typing-related issues might arise?
  - Have all possible service activities and service compositions been mapped out?
  - Have all compensation processes been fully tested?

Service testing (4)



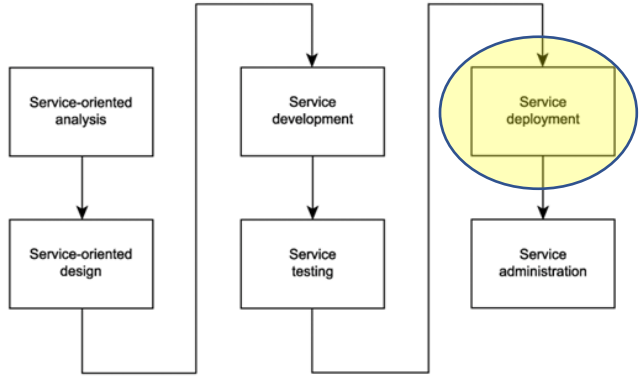
- Some of the key issues facing service testers:
  - What happens if exceptions occur within compensation processes?
  - Do all new services comply with existing design standards?
  - Do new services introduce custom SOAP headers? And, if yes, are all potential requestors (including intermediaries) required to do so, capable of understanding and processing them?
  - Do new services introduce functional or QoS requirements that the current architecture does not support?

Service deployment (1)



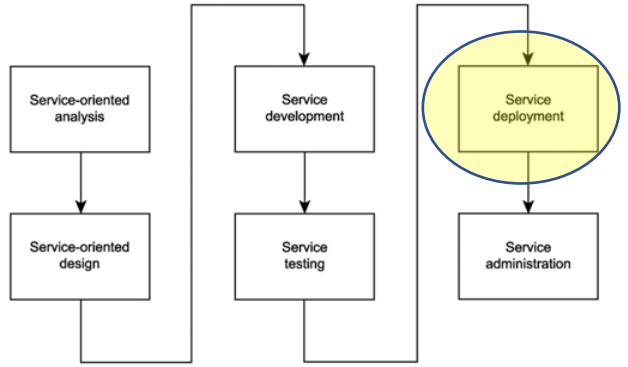
- Installing and configuring distributed components, service interfaces, and any associated middleware products onto production servers.

Service deployment (2)



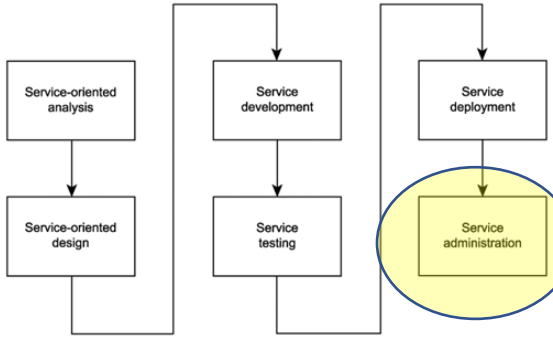
- Typical issues that arise during this phase include:
  - How will services be distributed?
  - Is the infrastructure adequate to fulfill the processing requirements of all services?
  - How will the introduction of new services affect existing services and applications?
  - How should services used by multiple solutions be positioned and deployed?
  - How will the introduction of any required middleware affect the existing environment?

Service deployment (3)



- Typical issues (cont):
  - Do these services introduce new versions of service descriptions that will need to be deployed alongside existing versions?
  - What security settings and accounts are required?
  - How will service pools be maintained to accommodate planned or unforeseen scalability requirements?
  - How will encapsulated legacy systems with performance or reliability limitations be maintained and monitored?

Service administration



- After services are deployed, standard application management issues come to the forefront.
- Issues frequently include:
  - How will service usage be monitored?
  - What form of version control will be used to manage service description documents?
  - How will messages be traced and managed?
  - How will performance bottlenecks be detected?

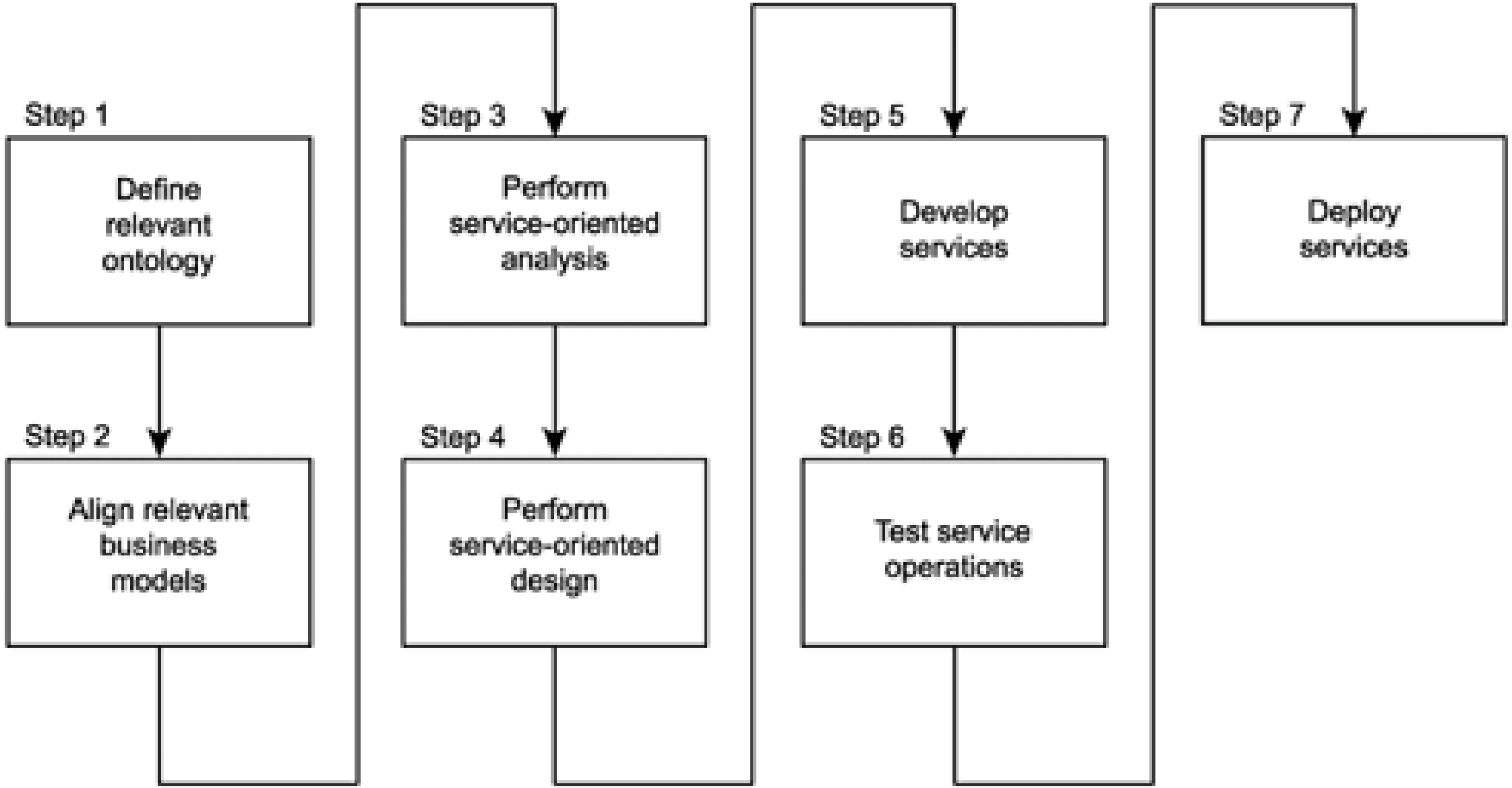
SOA Delivery Strategies

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

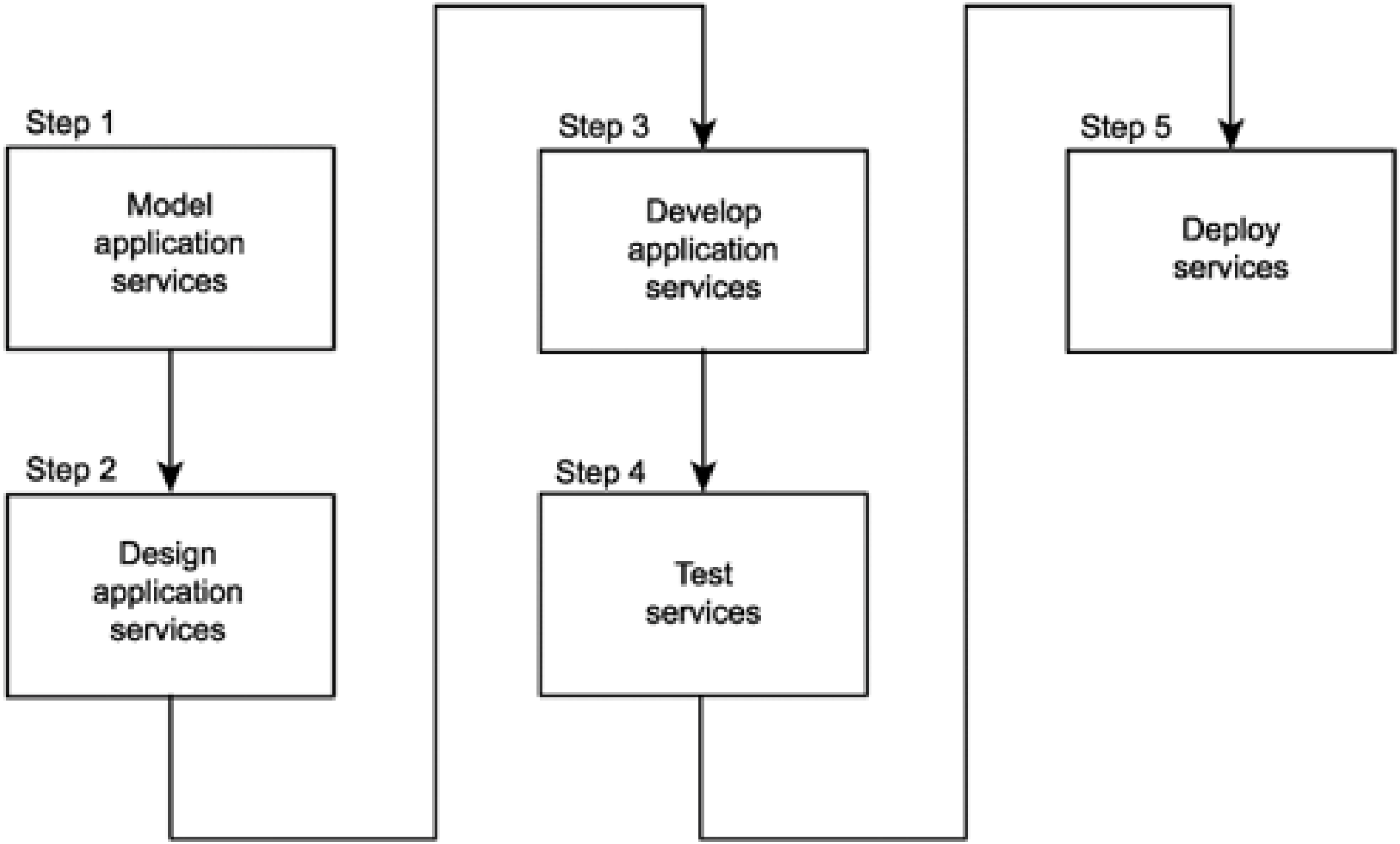

Step 1
Define relevant ontology
Step 2
Align relevant business models
Step 3
Perform service-oriented analysis
Step 4
Perform service-oriented design
Step 5
Develop services
Step 6
Test service operations
Step 7
Deploy services

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.

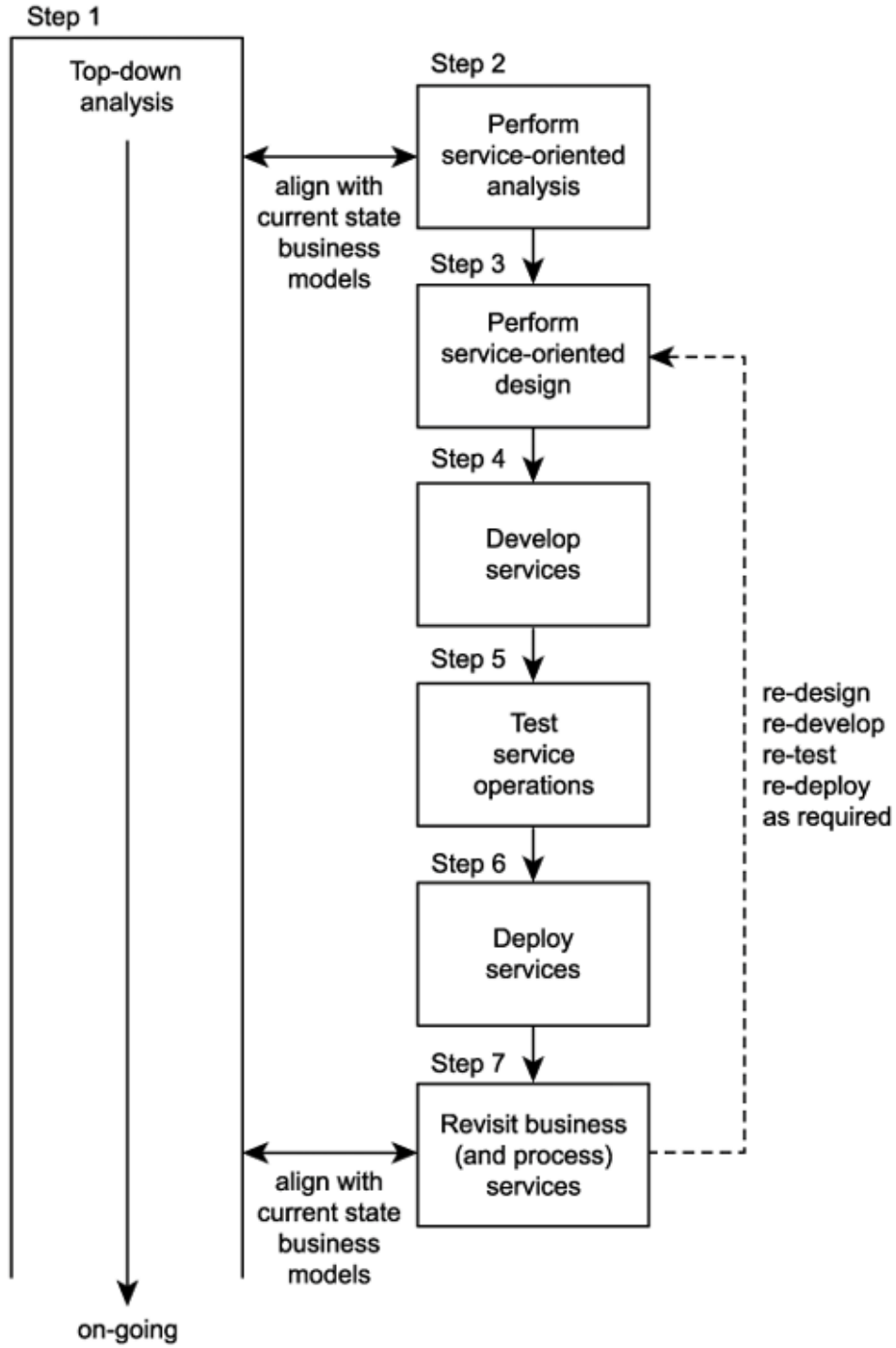

Step 1
Model application services
Step 2
Design application services
Step 3
Develop application services
Step 4
Test services
Step 5
Deploy services

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

RailCo case study review

- RailCo realised that it was loosing their clients to their competitor
- To prevent loosing their primary client, Transit Line Systems (TLS), they decided to build a pair of Web Services to participate in online transactions with TLS
  - RailCo-to-TLS Invoice Submission Process
  - TLS-to-RailCo Purchase Order Submission Process

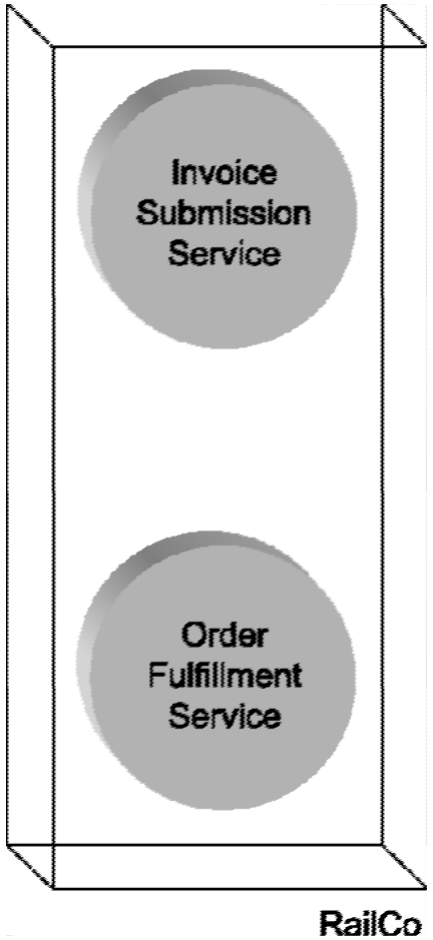


Figure 2.1: RailCo's initial set of Web services, designed only to allow RailCo to connect to TLS's B2B solution.

Multiple Choice(single) Points: 1

Setting

Based on the information given in the case study in Module 5, as well as in your homework assignment, RailCo constructed their services based on which approach?

A Bottom-up

B Top-down

C Agile

Submit

RailCo **bottom-up** Web services construction results

- The services were assembled quickly, to accommodate a single client

RailCo **bottom-up** Web services construction results

- The services were assembled quickly, to accommodate a single client
- The services were delivered to accommodate a specific need
  - fulfilled immediate business requirements

RailCo **bottom-up** Web services construction results

- The services were assembled quickly, to accommodate a single client
- The services were delivered to accommodate a specific need
  - fulfilled immediate business requirements
- Efficient

RailCo **bottom-up** Web services construction results

- The services were assembled quickly, to accommodate a single client
- The services were delivered to accommodate a specific need
  - fulfilled immediate business requirements
- Efficient
- Cost-effective

RailCo **bottom-up** Web services construction results

- The services were assembled quickly, to accommodate a single client
- The services were delivered to accommodate a specific need
  - fulfilled immediate business requirements
- Efficient
- Cost-effective
- Initially considered successful

However, very soon RailCo needed to respond to multiple changes

- TSL implemented a new policy that affected some of the purchase order business logic on RailCo’s end

However, very soon RailCo needed to respond to multiple changes

- TSL implemented a new policy that affected some of the purchase order business logic on RailCo's end
- RailCo's own internal business process underwent a change as a result of a module upgrade within their legacy accounting system
  - Affected low-level data access parameters processed by the RailCo Invoice Submission Service

However, very soon RailCo needed to respond to multiple changes

- TSL implemented a new policy that affected some of the purchase order business logic on RailCo's end
- RailCo's own internal business process underwent a change as a result of a module upgrade within their legacy accounting system
  - Affected low-level data access parameters processed by the RailCo Invoice Submission Service
- Other RailCo's legacy systems that interfaced with their Web services also underwent changes

However, very soon RailCo needed to respond to multiple changes

- TSL implemented a new policy that affected some of the purchase order business logic on RailCo's end
- RailCo's own internal business process underwent a change as a result of a module upgrade within their legacy accounting system
  - Affected low-level data access parameters processed by the RailCo Invoice Submission Service
- Other RailCo's legacy systems that interfaced with their Web services also underwent changes
- Numerous changes were made to existing automation processes

Discussion

- Each of the changes in numerous systems interfacing with RailCo’s Web services affected multiple services
- These services needed to be redeveloped
- What challenges do you see in redeveloping the Web services constructed based on bottom-up approach?

Open Question Points: 10

Setting

Each of the changes in numerous systems interfacing with RailCo’ s Web services affected multiple services
These services needed to be redeveloped
What challenges do you see in redeveloping the Web services constructed based on bottom-up approach?

Open Question is only supported on Version 2.0 or newer.

Answer

Redeveloping web services challenges

- A significant redevelopment effort

Redeveloping web services challenges

- A significant redevelopment effort
- There was no separation of business and application logic
  - Change impacted a broader part of their solution environment

Redeveloping web services challenges

- A significant redevelopment effort
- There was no separation of business and application logic
  - Change impacted a broader part of their solution environment
- None of the services were reusable
  - Less chance that the new requirements could be fulfilled by existing services

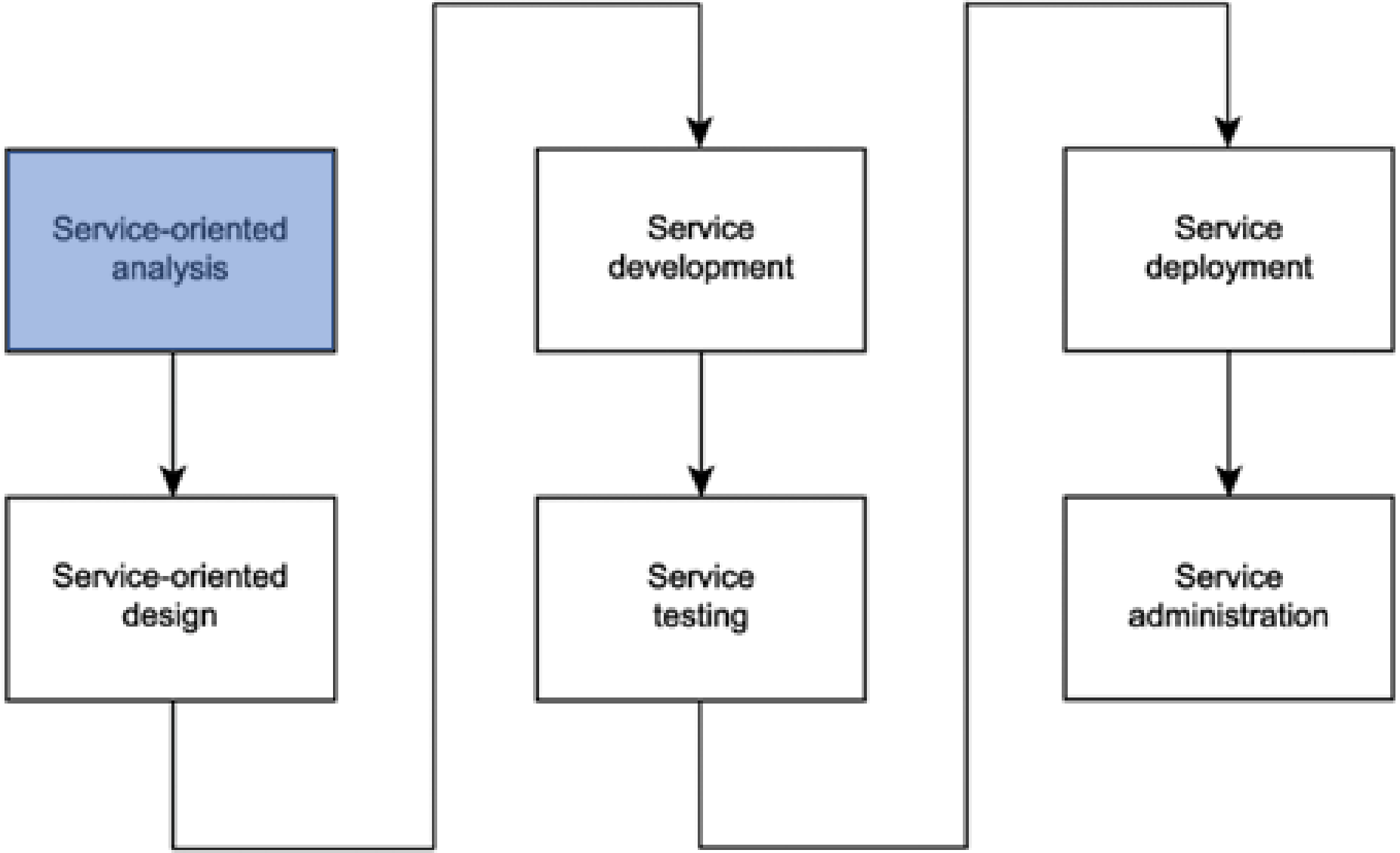
Redeveloping web services challenges

- A significant redevelopment effort
- There was no separation of business and application logic
  - Change impacted a broader part of their solution environment
- None of the services were reusable
  - Less chance that the new requirements could be fulfilled by existing services
- With each programming modification came the overhead
  - Testing and development phases

RailCo decides to start from scratch

- The agile delivery strategy is chosen
- Only application and business service layers will be modeled (cannot afford to invest in the middleware required to implement an orchestration layer)

SOA delivery lifecycle



Service-oriented analysis

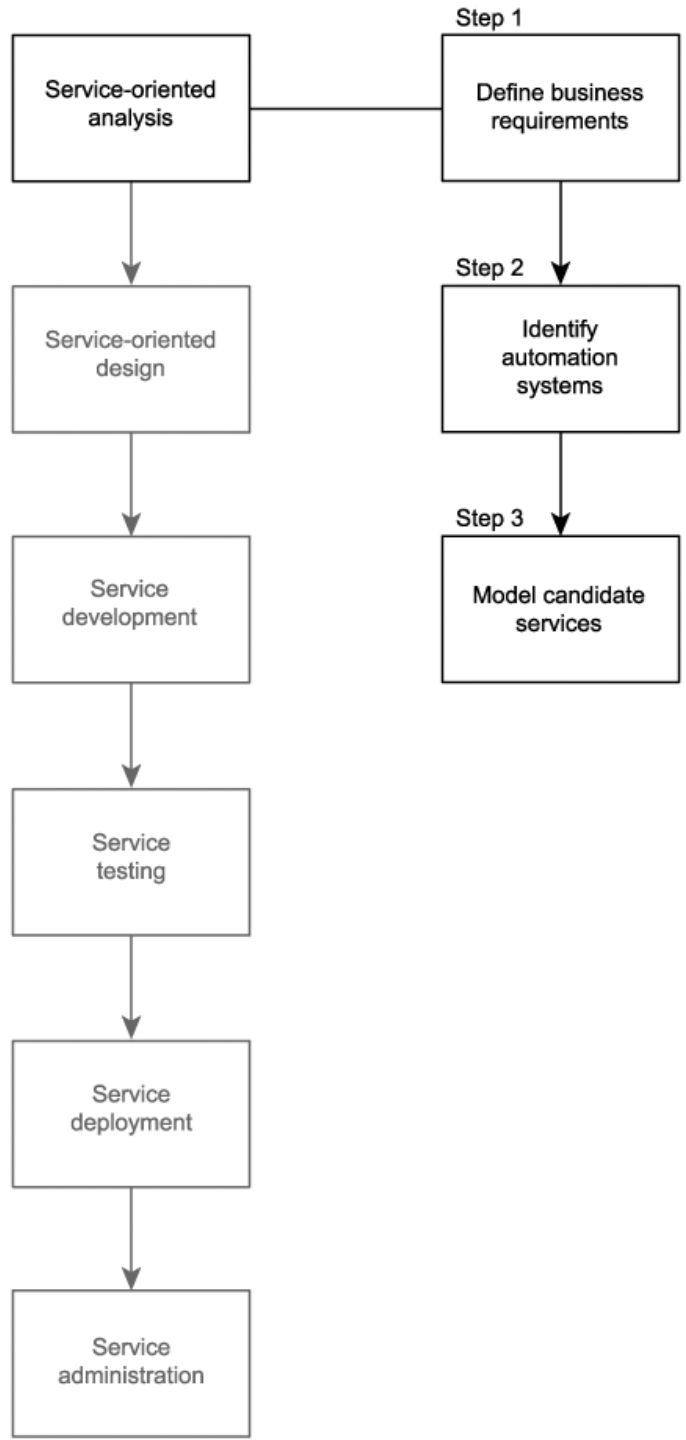
- The service-oriented analysis phase is probably the most important part of our SOA delivery lifecycle.

Service-oriented analysis objectives

- The primary questions addressed during this phase are:
  - What services need to be built?
  - What logic should be encapsulated by each service?

Service-oriented analysis goals

- Define a preliminary set of service operation candidates.
- Group service operation candidates into logical contexts. These contexts represent service candidates.
- Define preliminary service boundaries so that they do not overlap with any existing or planned services.
- Identify encapsulated logic with reuse potential.
- Ensure that the context of encapsulated logic is appropriate for its intended use.
- Define any known preliminary composition models.


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

Step 1: Define business automation requirements

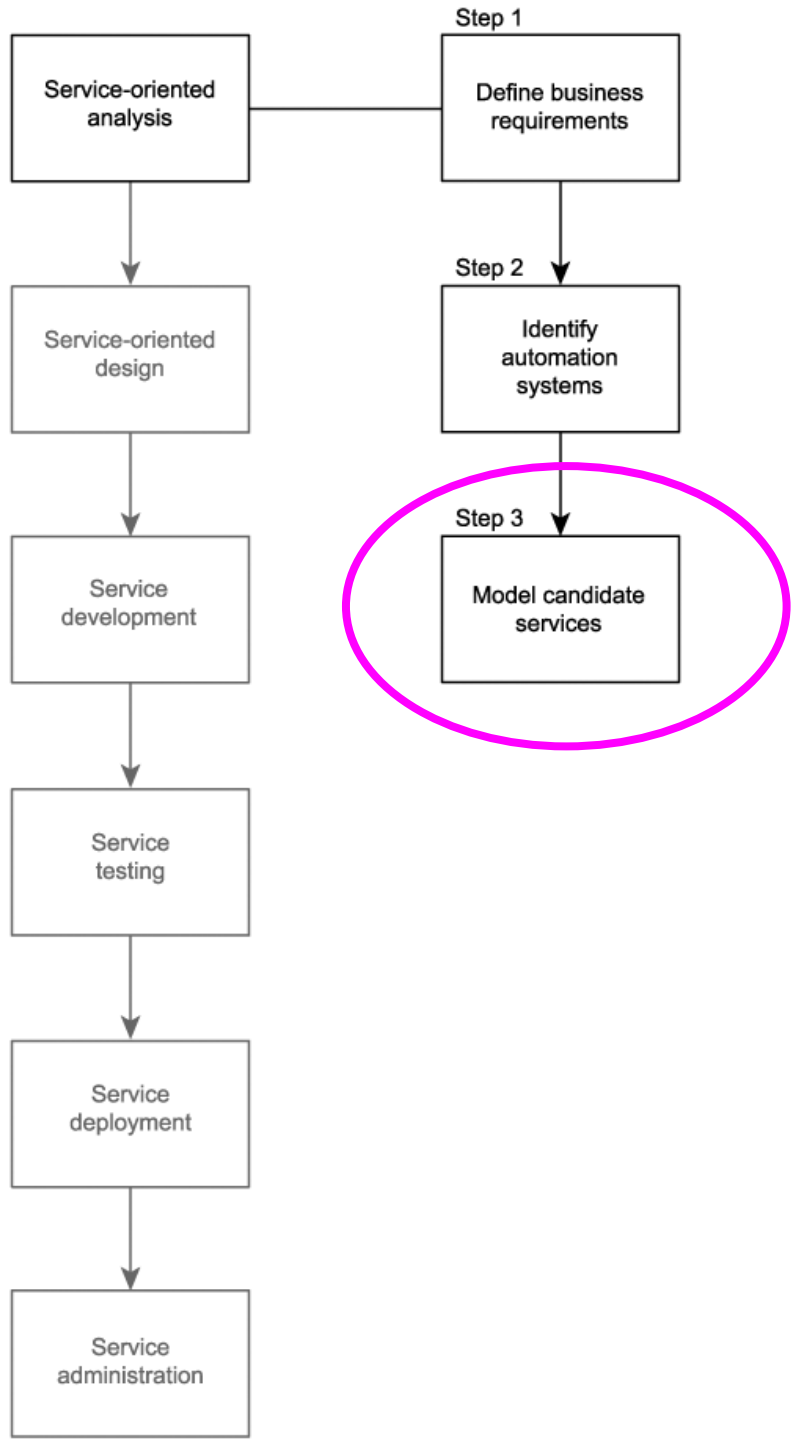
- Business requirements documentation is required for this analysis process to begin
- Only requirements related to the scope of a service-oriented solution should be considered
- This business process documentation will be used as the starting point of the service modeling process described in Step 3.

Step 2: Identify existing automation systems

- Gain an understanding of affected legacy environments
- Existing application logic that is already automating any of the requirements identified in Step 1 needs to be identified
- This information will be used to help identify application service candidates during the service modeling process described in Step 3

Step 3: Model candidate services

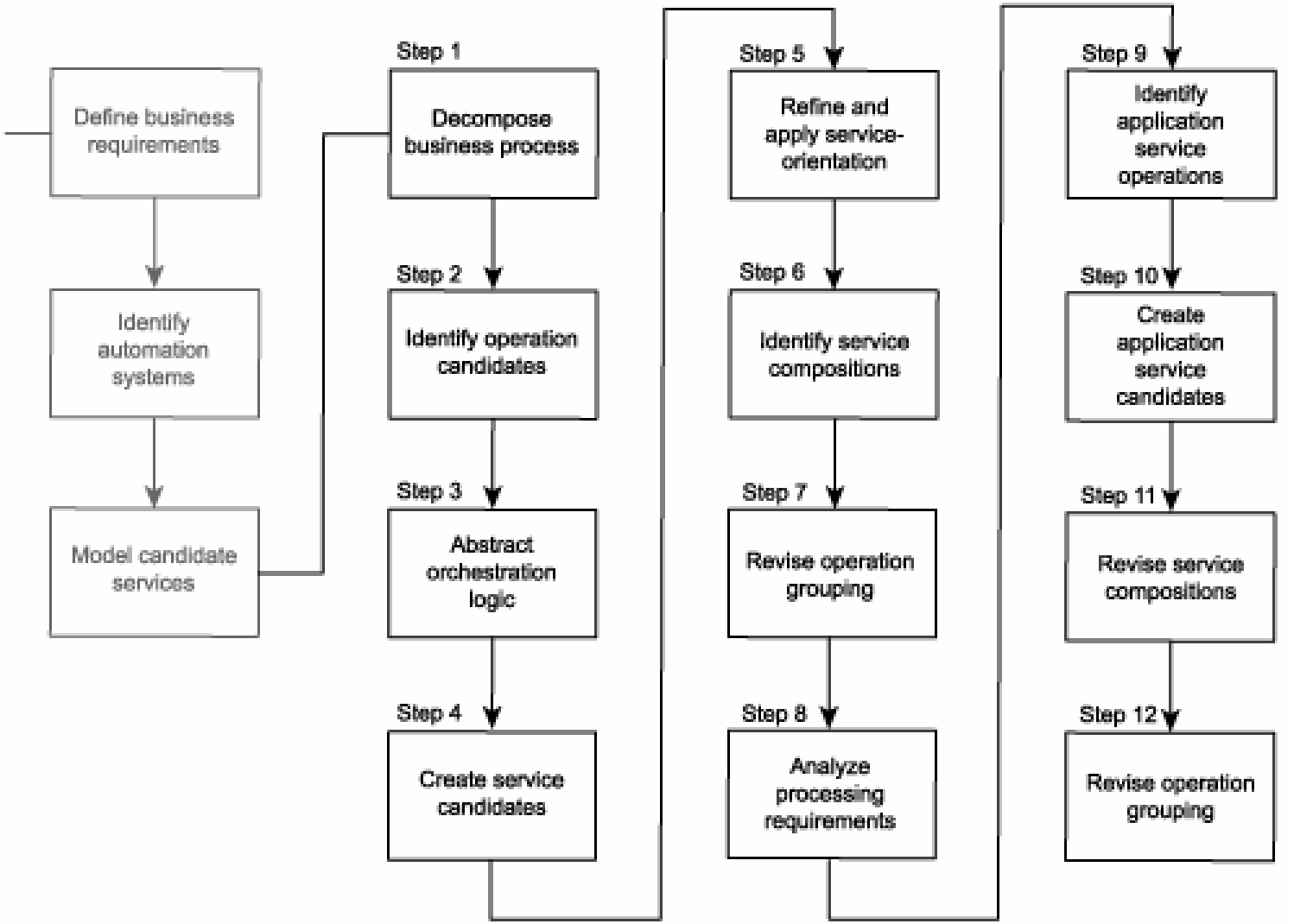
- Service operation candidates are identified and then grouped into a logical context
- These groups eventually take shape as service candidates
  - they are then further assembled into a tentative composite model representing the combined logic of the planned service-oriented application


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

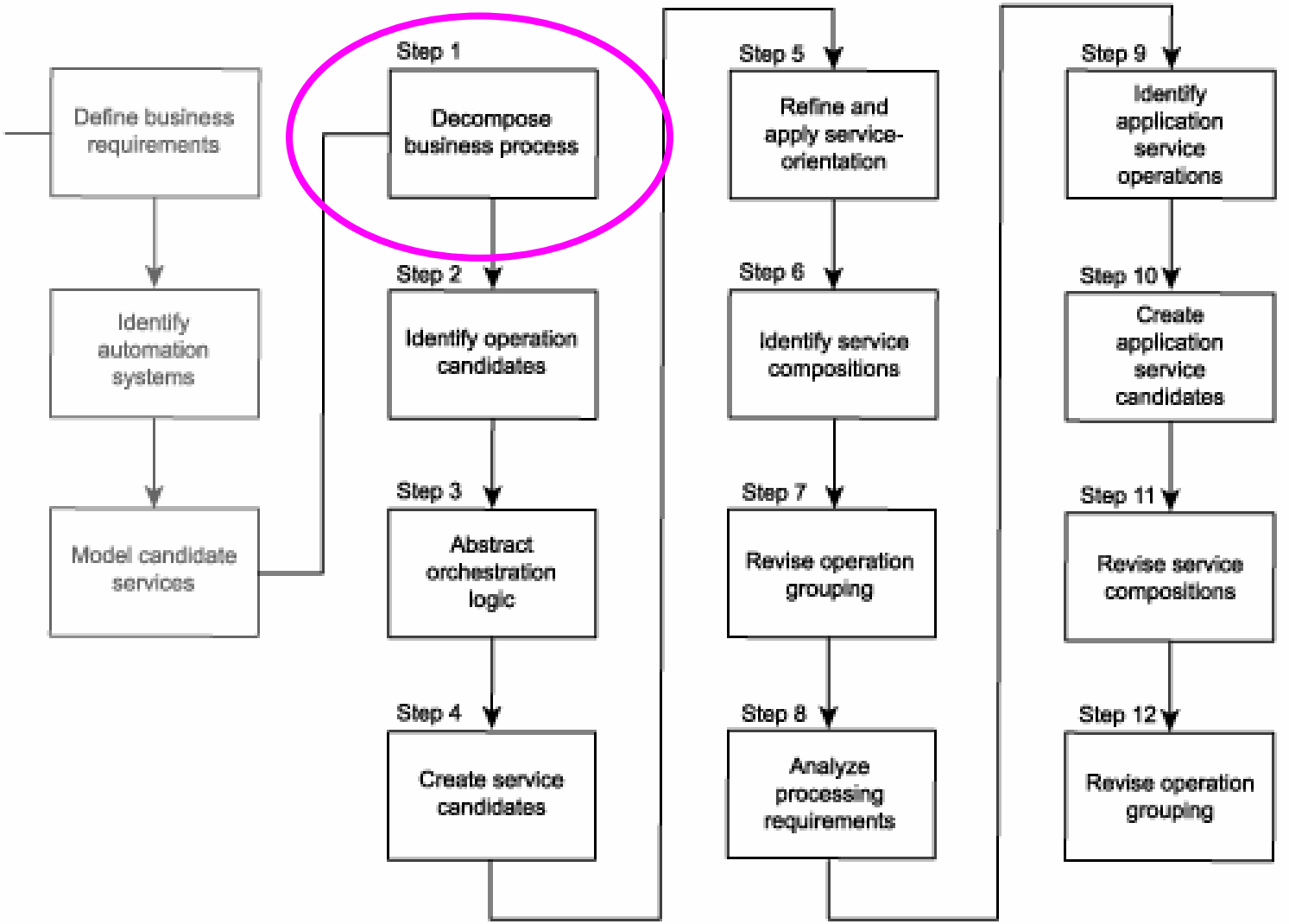
Service Modelling

- A service modeling process - organizing the information we gathered in Steps 1 and 2 of the service-oriented analysis process
- This process can be structured in many different ways
- We will provide some guidelines that can be used to customize the process to fit within organization's existing business analysis platforms and procedures

Common steps of modeling process



Common steps of modeling process



Step 1: Decompose the business process

- Take the documented business process and break it down into a series of granular process steps

Step 1 example

- We will show this step on the example of RailCo
- First, let’ have a look at their business processes

Invoice Submission Process

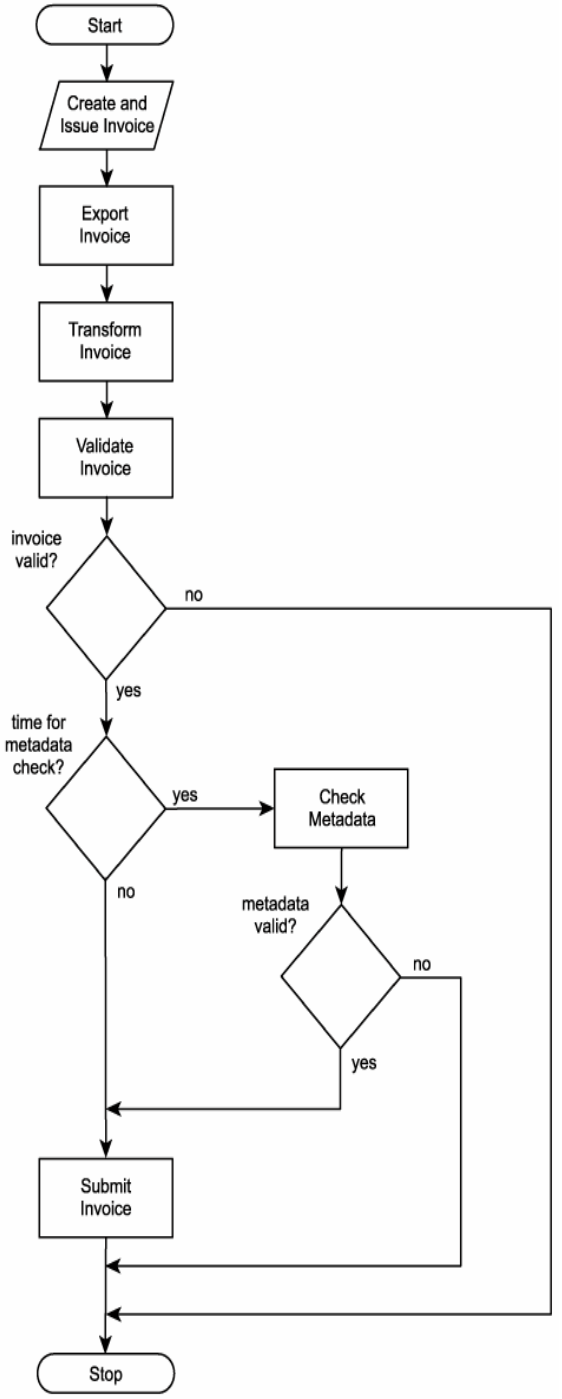
- Accounting clerk creates and issues an electronic invoice using the legacy accounting system.
- The save event triggers a custom script that exports an electronic copy of the invoice to a network folder.
- A custom developed component, which polls this folder at ten-minute intervals, picks up the document 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.

Step 1 example

- Then, we will take this business process and break it down into a series of granular process steps

Invoice Submission Process decomposition

- 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.



Open Question Points: 10

Setting

Perform the first step of modelling process (decompose the business process) based on RailCo's 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.

Open Question is only supported on Version 2.0 or newer.

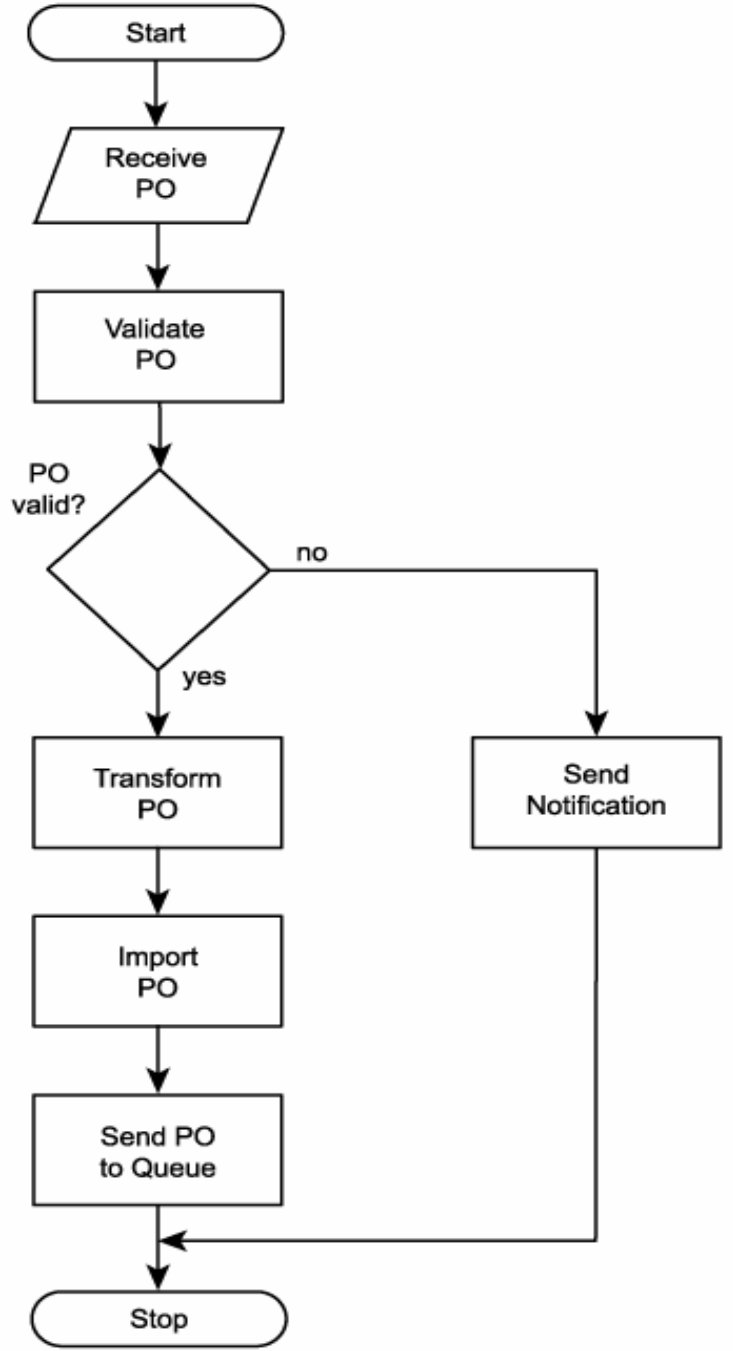
Answer

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.

Order Fulfilment Process decomposition

- 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.



Step 2: Identify business service operation candidates

- Some steps within a business process can be easily identified as not belonging to the potential logic that should be encapsulated by a service candidate
  - Manual process steps that cannot or should not be automated
  - Process steps performed by existing legacy logic for which service candidate encapsulation is not an option
- By filtering out these parts we are left with the processing steps most relevant to our service modeling process.

Step 2 example

- We will show this step on the example of RailCo
- First, recall their decomposed business processes

Invoice Submission Process:

- ~~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 Process

- 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.)

Step 2 RailCo example

- The result of this review is that only two processing steps were removed from the Invoice Submission Process and none from the Order Fulfillment Process.

Step 3: Abstract orchestration logic

- If the decision was made to not incorporate an orchestration service layer - then skip this step
- If it was decided to build an orchestration layer as part of the SOA, then we should identify the parts of the processing logic that this layer would abstract
- Potential types of logic suitable for this layer include:
  - business rules
  - conditional logic
  - exception logic
  - sequence logic

Step 4: Create business service candidates

- Review the processing steps that remain after we completed step 2 and determine one or more logical contexts with which these steps can be grouped.
- Each context represents a service candidate.
- In this step, we can also add additional service operation candidates not required by the current business process, but added to round out entity services with a complete set of reusable operations.

Step 4 example

- We will continue our RailCo case to illustrate this step

Example

- First, we will review the list of steps in both processes of RailCo (without those we crossed out)

Invoice Submission Process:

- ~~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 Process

- 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.)

Example

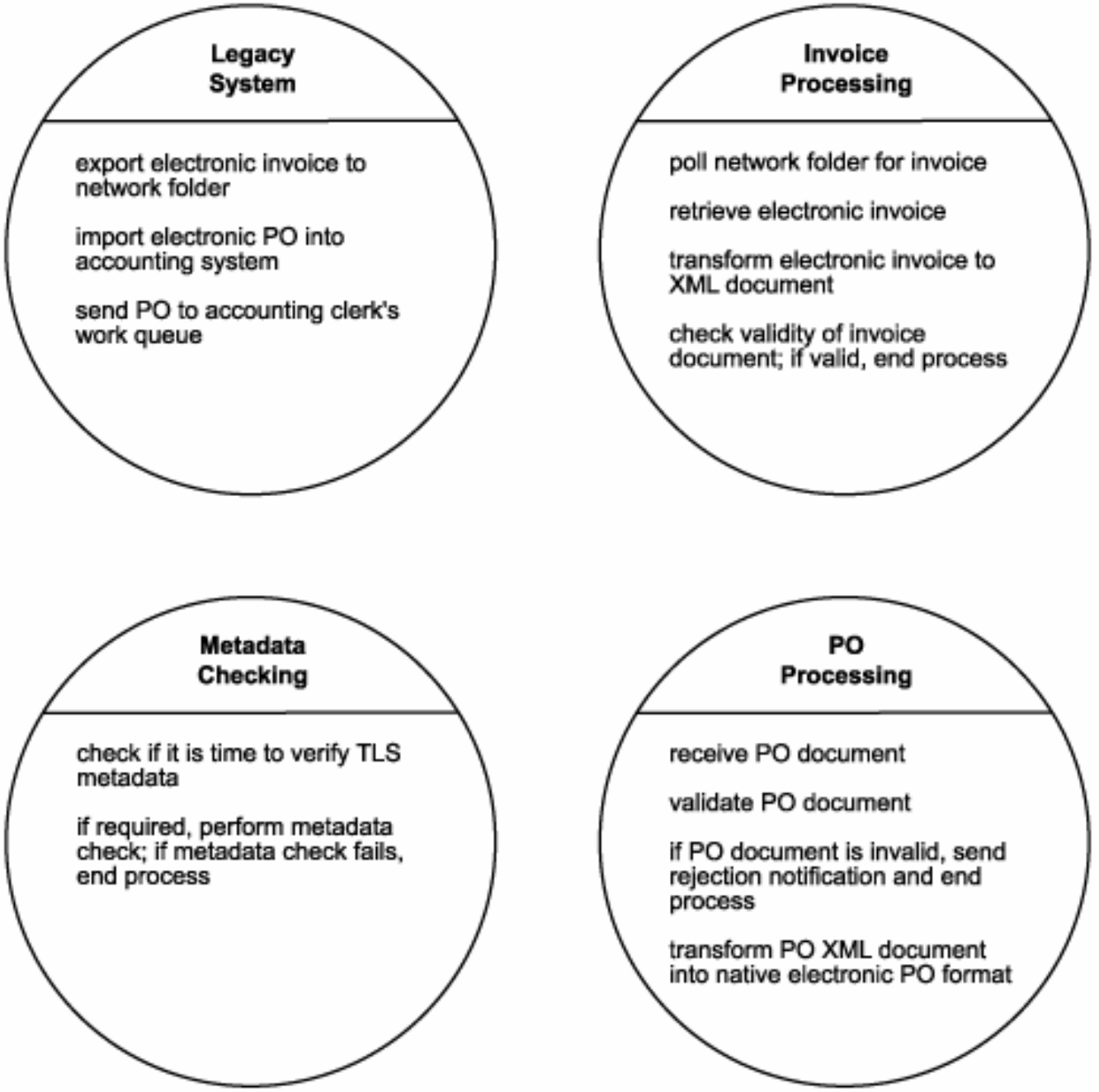
- Now, we need to group them into logical contexts
- What are the contexts we can group remaining services into?

Step 4 RailCo example

- Going through the RailCo steps we listed from both processes, here is how they could be grouped:
  - Legacy System Service
    - Export electronic invoice to network folder.
    - Import electronic PO into accounting system.
    - Send PO to accounting clerk's work queue.
  - Invoice Processing Service
    - Poll network folder for invoice.
    - Retrieve electronic invoice.
    - Transform electronic invoice to XML document.
    - Check validity of invoice document. If invalid, end process.
  - PO Processing Service
    - 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.
  - Metadata Checking Service
    - Check if it is time to verify TLS metadata.
    - If required, perform metadata check. If metadata check fails, end process.

Step 4 RailCo example (2)

- RailCo’s service candidates

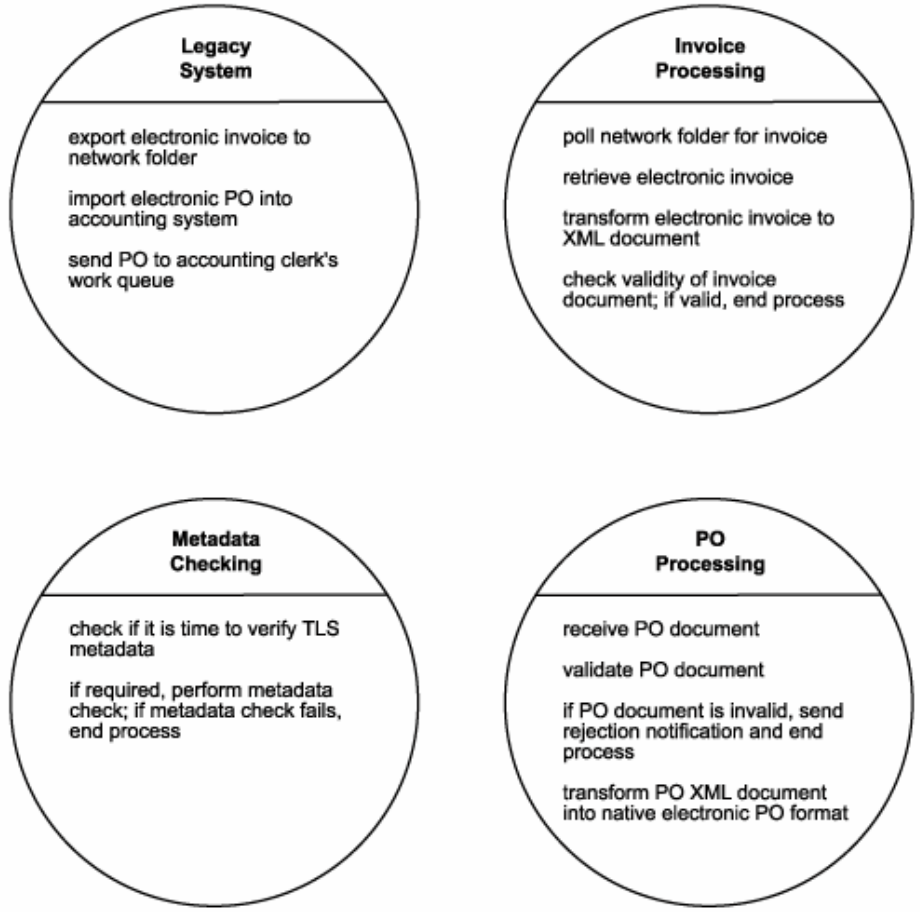


Step 5: Refine and apply principles of service-orientation

- 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



Let us analyse our current design of Legacy System Service

- Legacy System Service has at the moment 3 service operation candidates
  - Export electronic invoice to network folder.
  - Import electronic PO into accounting system.
  - Send PO to accounting clerk's work queue.

Activity

- Apply reusability and autonomy principles to adjust the service candidates in our initial design

Step 5 RailCo example – applying autonomy principle

- 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.

Open Question Points: 10

Setting

We apply the autonomy principle in our design:

1. We identified two operation candidates that are dependent on each other

   "Send PO to accounting clerk's work queue"

   "Import electronic PO into accounting system“

2. We decide to make an adjustment and combine them into one operation candidate

Suggest the name of the new operation candidate.

Open Question is only supported on Version 2.0 or newer.

Answer

Step 5 RailCo example – applying autonomy principle

- 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

- Further the "Import electronic PO into accounting system" action is performed automatically by a macro added to the legacy accounting system. It is therefore not required as part of our service candidate.

Step 5 RailCo example – applying reusability principle

- This leaves us with a single operation candidate “Export electronic invoice to network folder” that we would like to make more reusable by allowing it to handle different types of documents.

Open Question Points: 10

Setting

Make “Export electronic invoice to network folder” more reusable by designing it in more generic way *(hint: give it a more generic name)*

Suggest the name of the new operation candidate.

Open Question is only supported on Version 2.0 or newer.

Answer

Step 5 RailCo example – applying reusability principle

- This leaves us with a single operation candidate "Export electronic invoice to network folder" 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

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

Activity

- 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?

Making adjustments in Invoice Processing Service

- Invoice Processing Service has 4 operation candidates at the moment
  - Poll network folder for invoice.
  - Retrieve electronic invoice.
  - Transform electronic invoice to XML document.
  - Check validity of invoice document. If invalid, end process.

Open Question Points: 10

Setting

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?

Open Question is only supported on Version 2.0 or newer.

Answer

Step 5 RailCo example

- 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”

Making adjustments in Invoice Processing Service

- Invoice Processing Service has 4 operation candidates at the moment
  - ~~Poll network folder for invoice~~. -→ poll folder for new documents
  - Retrieve electronic invoice.
  - Transform electronic invoice to XML document.
  - Check validity of invoice document. If invalid, end process.

Step 5 RailCo example

- 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.

Making adjustments in Invoice Processing Service

- Invoice Processing Service has 4 operation candidates at the moment
  - ~~Poll network folder for invoice~~. -→ ~~poll folder for new documents~~
  - Retrieve electronic invoice.
  - Transform electronic invoice to XML document.
  - Check validity of invoice document. If invalid, end process.

- Polling Notification Service:
  - Poll folder for new documents.
  - If documents arrive for which there are subscribers, issue notifications.

Step 5 RailCo example

- 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.

Making adjustments in Invoice Processing Service

- Invoice Processing Service has 4 operation candidates at the moment
  - ~~Poll network folder for invoice~~. -→ ~~poll folder for new documents~~
  - ~~Retrieve electronic invoice.~~
  - ~~Transform electronic invoice to XML document.~~
  - ~~Check validity of invoice document. If invalid, end process.~~
  - Retrieve and transform invoice document.
- Polling Notification Service:
  - Poll folder for new documents.
  - If documents arrive for which there are subscribers, issue notifications.

Adjusting PO Processing Service

- PO Processing Service
  - 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.

Step 5 RailCo example

- 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.

Adjusting PO Processing Service

- PO Processing Service
  - ~~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.

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.“

Adjusting PO Processing Service

- PO Processing Service
  - ~~Receive PO document.~~
  - ~~Validate PO document.~~
  - ~~If PO document is invalid, send rejection notification and end process.~~
  - Validate PO document and send rejection notification, if required
  - Transform PO XML document into native electronic PO format.

Multiple Choice(single) Points: 1

Setting

In our previous exercise, we have detected a direct dependency between the two operation candidates and combined them into one candidate. Which service design principle have we applied?

A Autonomy

B Reusability

C Discoverability

Submit

- We will continue this topic next week