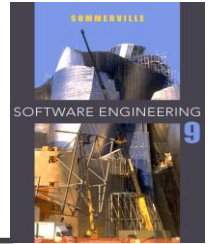


---

# Service-oriented Architecture

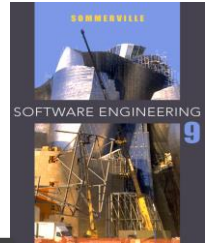
# Topics covered

---



- ✧ Services as reusable components
- ✧ Service engineering
- ✧ Software development with services

# Web services



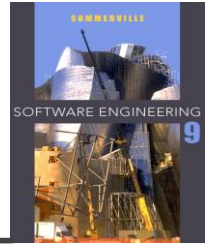
- ✧ A web service is an instance of a more general notion of a service:

*“an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production”.*

- ✧ The essence of a service, therefore, is that the provision of the service is independent of the application using the service.
- ✧ Service providers can develop specialized services and offer these to a range of service users from different organizations.

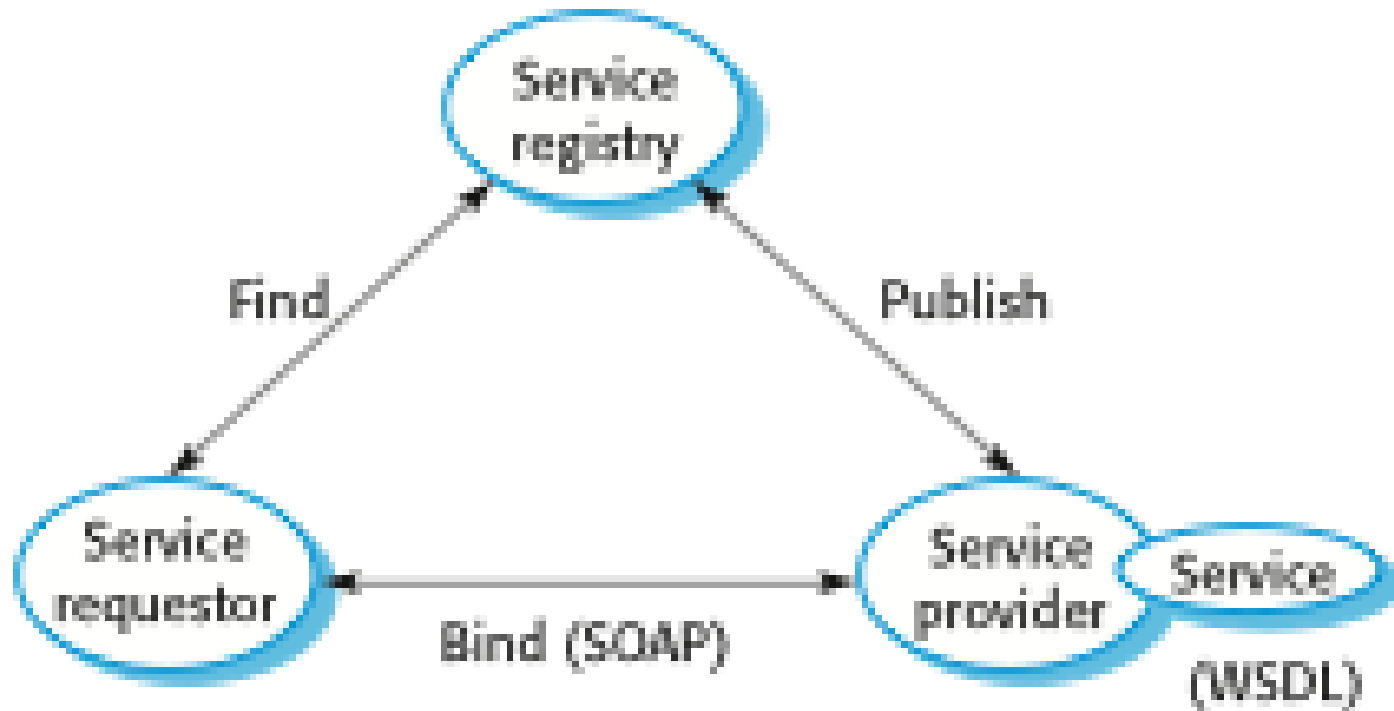
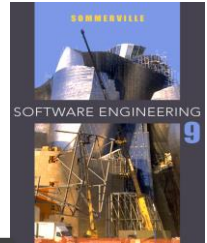
# Service-oriented architectures

---



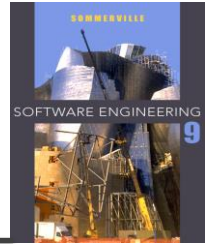
- ✧ A means of developing distributed systems where the components are stand-alone services
- ✧ Services may execute on different computers from different service providers
- ✧ Standard protocols have been developed to support service communication and information exchange

# Service-oriented architecture

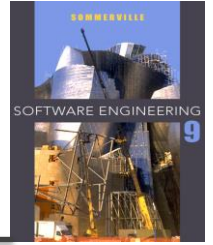


# Benefits of SOA

---



- ✧ Services can be provided locally or outsourced to external providers
- ✧ Services are language-independent
- ✧ Investment in legacy systems can be preserved
- ✧ Inter-organisational computing is facilitated through simplified information exchange



# Key standards

---

## ✧ SOAP

- A message exchange standard that supports service communication

## ✧ WSDL (Web Service Definition Language)

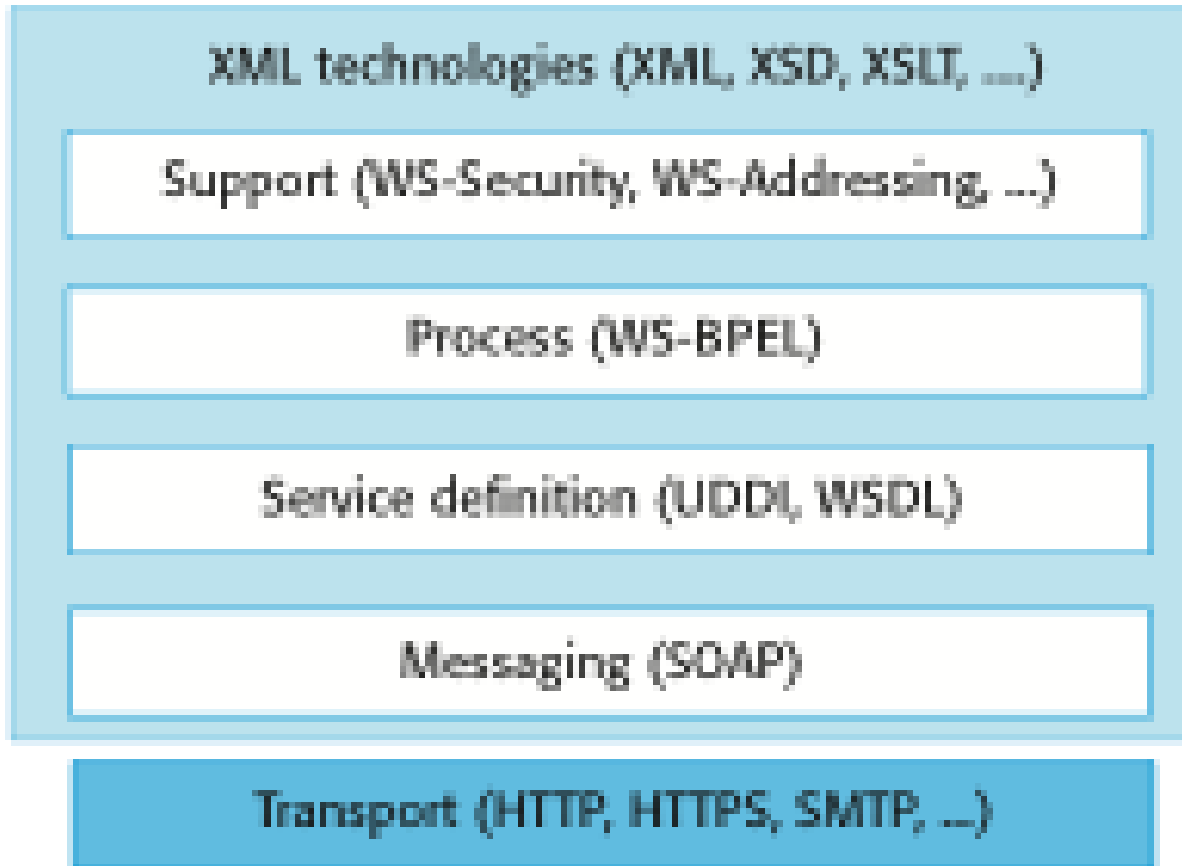
- This standard allows a service interface and its bindings to be defined

## ✧ WS-BPEL

- A standard for workflow languages used to define service composition

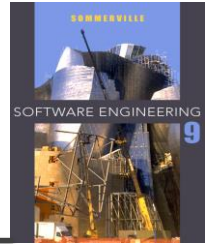
# Web service standards

---



# RESTful web services

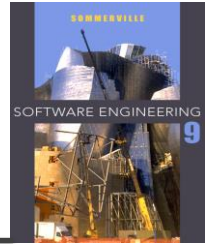
---



- ✧ Current web services standards have been criticized as ‘heavyweight’ standards that are over-general and inefficient.
- ✧ REST (REpresentational State Transfer) is an architectural style based on transferring representations of resources from a server to a client.
- ✧ This style underlies the web as a whole and is simpler than SOAP/WSDL for implementing web services.
- ✧ RESTFul services involve a lower overhead than so-called ‘big web services’ and are used by many organizations implementing service-based systems that do not rely on externally-provided services.

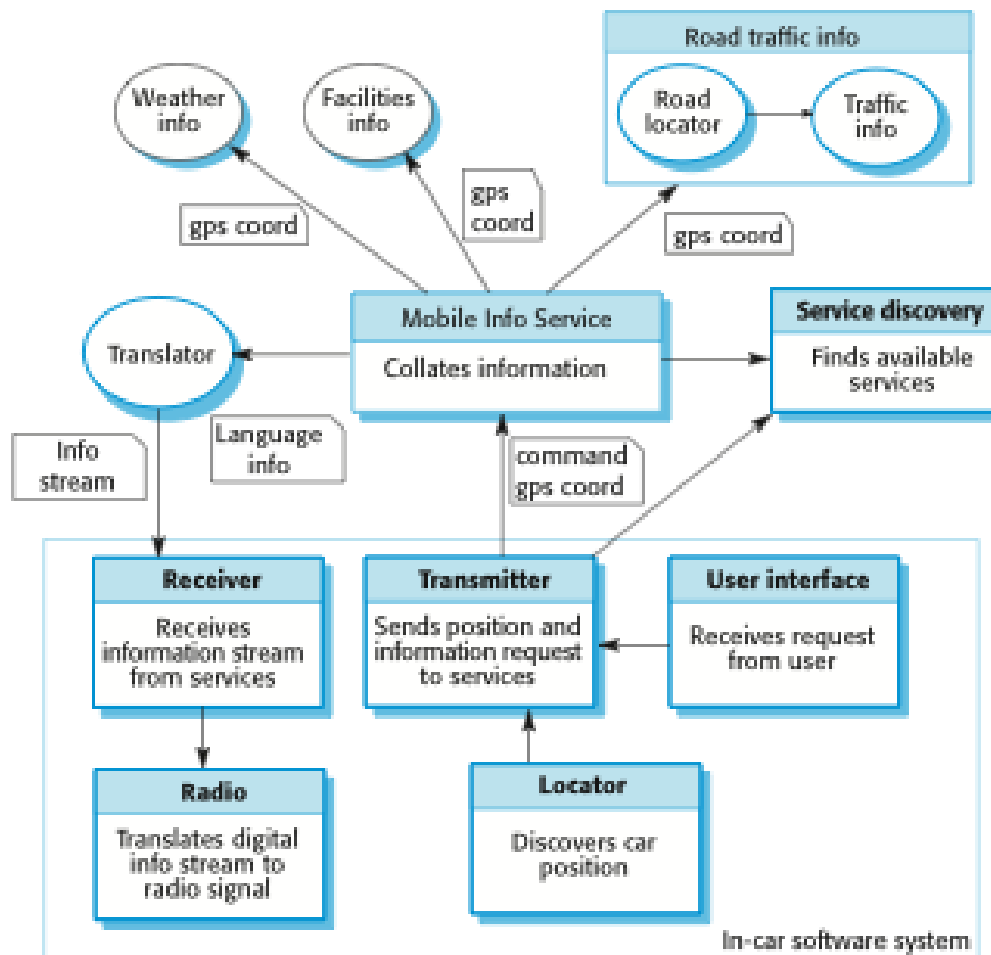
# Services scenario

---



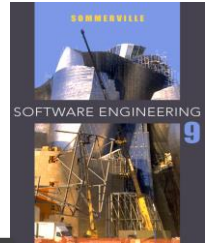
- ✧ An in-car information system provides drivers with information on weather, road traffic conditions, local information etc. This is linked to car radio so that information is delivered as a signal on a specific radio channel.
- ✧ The car is equipped with GPS receiver to discover its position and, based on that position, the system accesses a range of information services. Information may be delivered in the driver's specified language.

# A service-based, in-car information system



# Advantage of SOA for this application

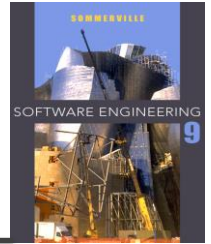
---



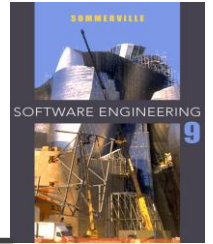
- ✧ It is not necessary to decide when the system is programmed or deployed what service provider should be used or what specific services should be accessed.
  - As the car moves around, the in-car software uses the service discovery service to find the most appropriate information service and binds to that.
  - Because of the use of a translation service, it can move across borders and therefore make local information available to people who don't speak the local language.

# Service-oriented software engineering

---



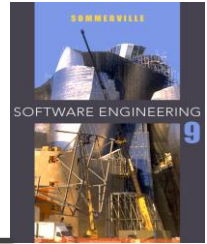
- ✧ Existing approaches to software engineering have to evolve to reflect the service-oriented approach to software development
  - Service engineering. The development of dependable, reusable services
    - Software development for reuse
  - Software development with services. The development of dependable software where services are the fundamental components
    - Software development with reuse



# Services as reusable components

---

- ✧ A service can be defined as:
  - *A loosely-coupled, reusable software component that encapsulates discrete functionality which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet and XML-based protocols*
- ✧ A critical distinction between a service and a component as defined in CBSE is that services are independent
  - Services do not have a 'requires' interface
  - Services rely on message-based communication with messages expressed in XML

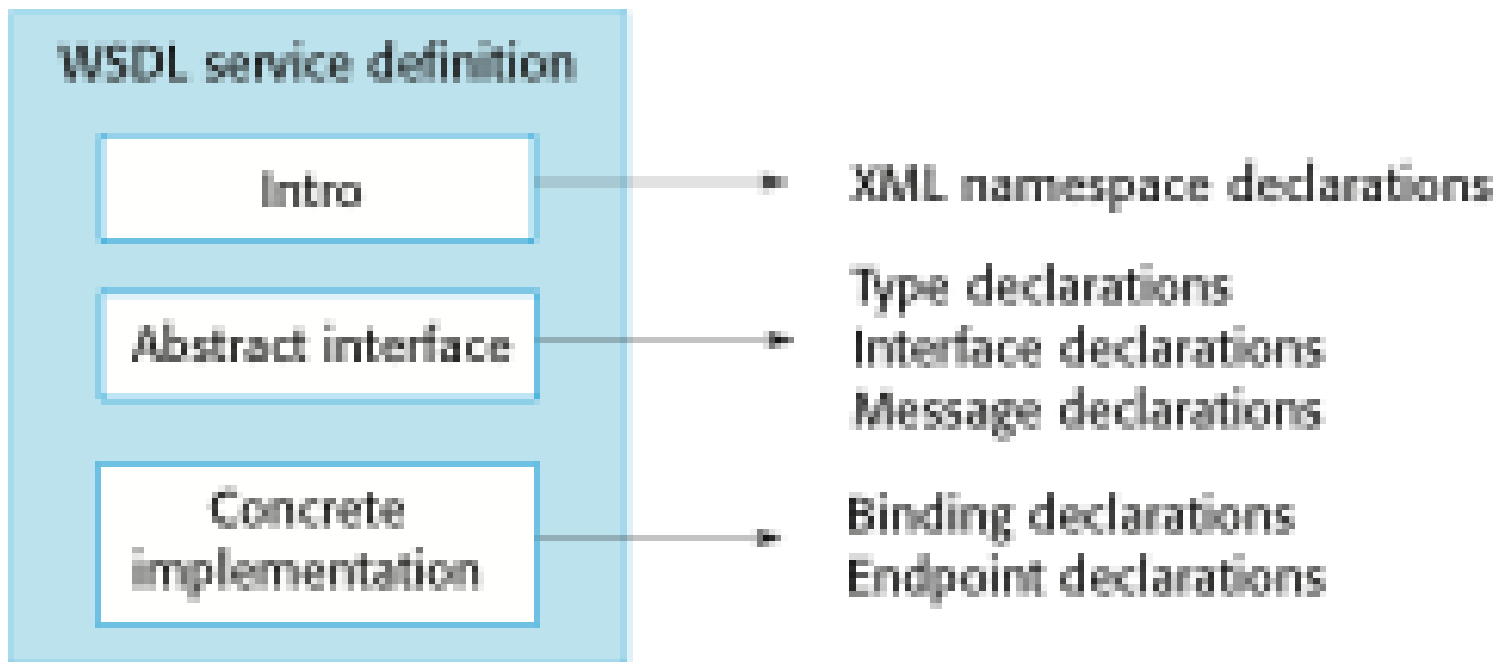


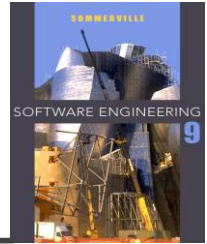
# Web service description language

---

- ✧ The service interface is defined in a service description expressed in WSDL (Web Service Description Language).
- ✧ The WSDL specification defines
  - What operations the service supports and the format of the messages that are sent and received by the service
  - How the service is accessed - that is, the binding maps the abstract interface onto a concrete set of protocols
  - Where the service is located. This is usually expressed as a URI (Universal Resource Identifier)

# Organization of a WSDL specification

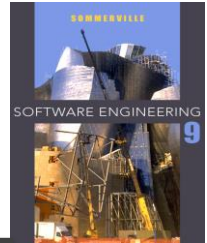




# WSDL specification components

---

- ✧ The ‘what’ part of a WSDL document, called an interface, specifies what operations the service supports, and defines the format of the messages that are sent and received by the service.
- ✧ The ‘how’ part of a WSDL document, called a binding, maps the abstract interface to a concrete set of protocols. The binding specifies the technical details of how to communicate with a Web service.
- ✧ The ‘where’ part of a WSDL document describes the location of a specific Web service implementation (its endpoint).



# Part of a WSDL description for a web service

*Define some of the types used. Assume that the namespace prefixes 'ws' refers to the namespace URI for XML schemas and the namespace prefix associated with this definition is weathns.*

```
<types>
  <xs: schema targetNamespace = "http://.../weathns"
    xmlns: weathns = "http://.../weathns" >
    <xs:element name = "PlaceAndDate" type = "pdrec" />
    <xs:element name = "MaxMinTemp" type = "mmtrec" />
    <xs: element name = "InDataFault" type = "errmess" />

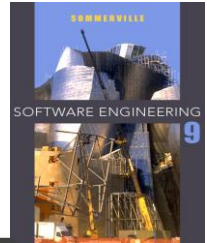
    <xs: complexType name = "pdrec"
      <xs: sequence>
        <xs:element name = "town" type = "xs:string"/>
        <xs:element name = "country" type = "xs:string"/>
        <xs:element name = "day" type = "xs:date" />
      </xs:complexType>

      Definitions of MaxMinType and InDataFault here

    </schema>
</types>
```

# Part of a WSDL description for a web service

---

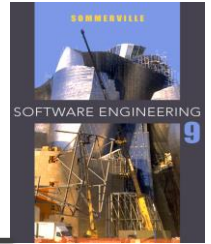


*Now define the interface and its operations. In this case, there is only a single operation to return maximum and minimum temperatures.*

```
<interface name = "weatherInfo" >
  <operation name = "getMaxMinTemps" pattern = "wsdl:ns: in-out">
    <input messageLabel = "In" element = "weathns: PlaceAndDate" />
    <output messageLabel = "Out" element = "weathns:MaxMinTemp" />
    <outfault messageLabel = "Out" element = "weathns:InDataFault" />
  </operation>
</interface>
```

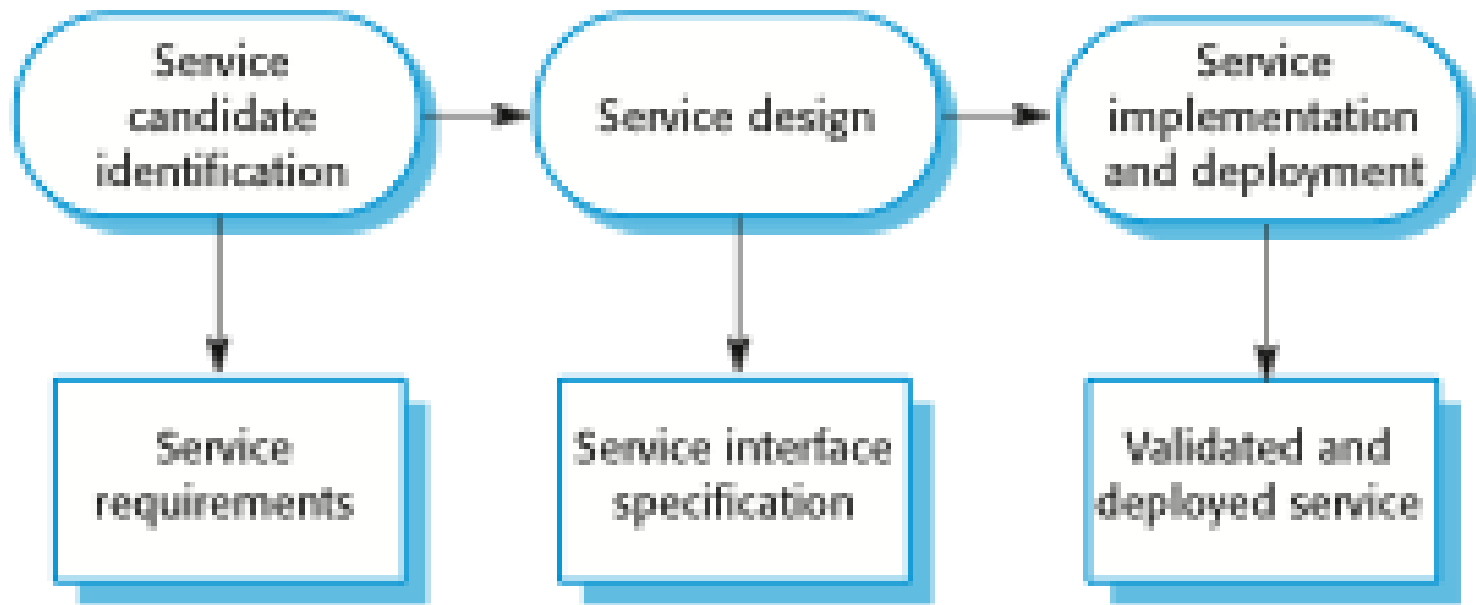
# Service engineering

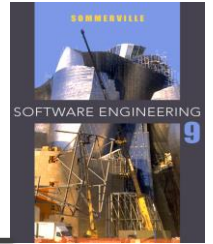
---



- ✧ The process of developing services for reuse in service-oriented applications
- ✧ The service has to be designed as a reusable abstraction that can be used in different systems.
- ✧ Generally useful functionality associated with that abstraction must be designed and the service must be robust and reliable.
- ✧ The service must be documented so that it can be discovered and understood by potential users.

# The service engineering process





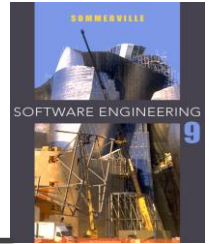
# Stages of service engineering

---

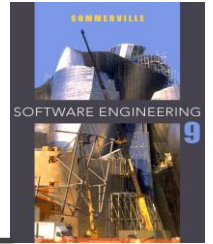
- ✧ Service candidate identification, where you identify possible services that might be implemented and define the service requirements.
- ✧ Service design, where you design the logical and WSDL service interfaces.
- ✧ Service implementation and deployment, where you implement and test the service and make it available for use.

# Service candidate identification

---



- ✧ Services should support business processes.
- ✧ Service candidate identification involves understanding an organization's business processes to decide which reusable services could support these processes.
- ✧ Three fundamental types of service
  - Utility services that implement general functionality used by different business processes.
  - Business services that are associated with a specific business function e.g., in a university, student registration.
  - Coordination services that support composite processes such as ordering.

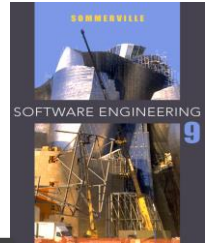


# Task and entity-oriented services

---

- ✧ Task-oriented services are those associated with some activity.
- ✧ Entity-oriented services are like objects. They are associated with a business entity such as a job application form.
- ✧ Utility or business services may be entity- or task-oriented, coordination services are always task-oriented.

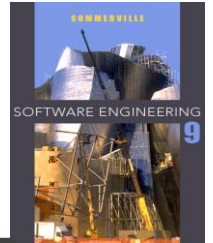
# Service classification



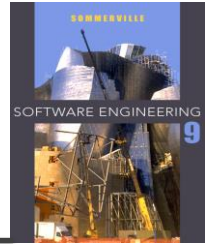
	Utility	Business	Coordination
Task	Currency converter Employee locator	Validate claim form Check credit rating	Process expense claim Pay external supplier
Entity	Document style checker Web form to XML converter	Expenses form Student application form	

# Key points

---



- ✧ Service-oriented software engineering is based on the notion that programs can be constructed by composing independent services which encapsulate reusable functionality.
- ✧ Service interfaces are defined in WSDL. A WSDL specification includes a definition of the interface types and operations, the binding protocol used by the service and the service location.
- ✧ Services may be classified as utility services, business services or coordination services.



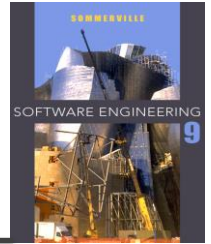
---

# Chapter 19 – Service-oriented Architecture

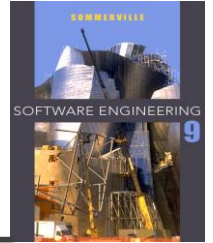
## Lecture 2

# Service identification

---



- ✧ Is the service associated with a single logical entity used in different business processes?
- ✧ Is the task one that is carried out by different people in the organisation?
- ✧ Is the service independent?
- ✧ Does the service have to maintain state? Is a database required?
- ✧ Could the service be used by clients outside the organisation?
- ✧ Are different users of the service likely to have different non-functional requirements?



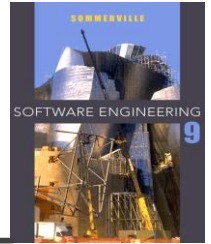
# Service identification example

---

- ✧ A large company, which sells computer equipment, has arranged special prices for approved configurations for some customers.
- ✧ To facilitate automated ordering, the company wishes to produce a catalog service that will allow customers to select the equipment that they need.
- ✧ Unlike a consumer catalog, orders are not placed directly through a catalog interface. Instead, goods are ordered through the web-based procurement system of each company that accesses the catalog as a web service.
- ✧ Most companies have their own budgeting and approval procedures for orders and their own ordering process must be followed when an order is placed.

# Catalog services

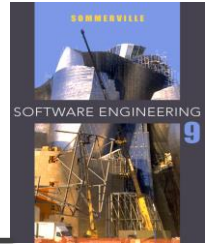
---



- ✧ Created by a supplier to show which good can be ordered from them by other companies
- ✧ Service requirements
  - Specific version of catalogue should be created for each client
  - Catalogue shall be downloadable
  - The specification and prices of up to 6 items may be compared
  - Browsing and searching facilities shall be provided
  - A function shall be provided that allows the delivery date for ordered items to be predicted
  - Virtual orders shall be supported which reserve the goods for 48 hours to allow a company order to be placed

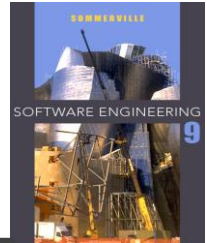
# Catalogue - Non-functional requirements

---



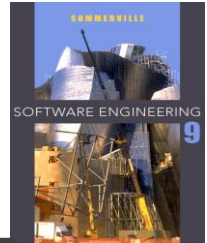
- ✧ Access shall be restricted to employees of accredited organisations
- ✧ Prices and configurations offered to each organisation shall be confidential
- ✧ The catalogue shall be available from 0700 to 1100
- ✧ The catalogue shall be able to process up to 10 requests per second

# Functional descriptions of catalog service operations



Operation	Description
MakeCatalog	Creates a version of the catalog tailored for a specific customer. Includes an optional parameter to create a downloadable PDF version of the catalog.
Compare	Provides a comparison of up to six characteristics (e.g., price, dimensions, processor speed, etc.) of up to four catalog items.
Lookup	Displays all of the data associated with a specified catalog item.

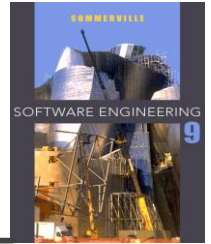
# Functional descriptions of catalog service operations



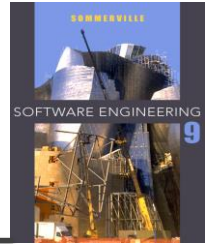
Operation	Description
Search	This operation takes a logical expression and searches the catalog according to that expression. It displays a list of all items that match the search expression.
CheckDelivery	Returns the predicted delivery date for an item if ordered that day.
MakeVirtualOrder	Reserves the number of items to be ordered by a customer and provides item information for the customer's own procurement system.

# Service interface design

---



- ✧ Involves thinking about the operations associated with the service and the messages exchanged
- ✧ The number of messages exchanged to complete a service request should normally be minimised.
- ✧ Service state information may have to be included in messages



# Interface design stages

---

## ✧ Logical interface design

- Starts with the service requirements and defines the operation names and parameters associated with the service. Exceptions should also be defined

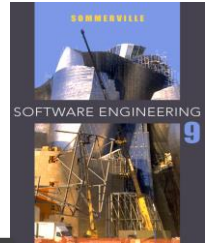
## ✧ Message design

- Design the structure and organisation of the input and output messages. Notations such as the UML are a more abstract representation than XML

## ✧ WSDL description

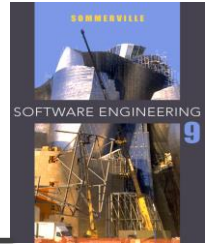
- The logical specification is converted to a WSDL description

# Catalog interface design



Operation	Inputs	Outputs	Exceptions
MakeCatalog	<i>mcIn</i> Company id PDF-flag	<i>mcOut</i> URL of the catalog for that company	<i>mcFault</i> Invalid company id
Compare	<i>compln</i> Company id Entry attribute (up to 6) Catalog number (up to 4)	<i>compOut</i> URL of page showing comparison table	<i>compFault</i> Invalid company id Invalid catalog number Unknown attribute
Lookup	<i>lookIn</i> Company id Catalog number	<i>lookOut</i> URL of page with the item information	<i>lookFault</i> Invalid company id Invalid catalog number

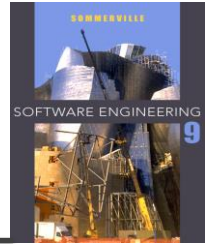
# Catalog interface design



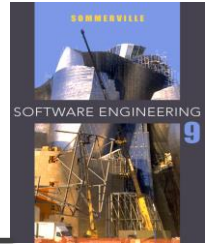
Operation	Inputs	Outputs	Exceptions
Search	<i>searchIn</i> Company id Search string	<i>searchOut</i> URL of web page with search results	<i>searchFault</i> Invalid company id Badly formed search string
CheckDelivery	<i>gdIn</i> Company id Catalog number Number of items required	<i>gdOut</i> Catalog number Expected delivery date	<i>gdFault</i> Invalid company id Invalid catalog number No availability Zero items requested
PlaceOrder	<i>poIn</i> Company id Number of items required Catalog number	<i>poOut</i> Catalog number Number of items required Predicted delivery date Unit price estimate Total price estimate	<i>poFault</i> Invalid company id Invalid catalog number Zero items requested

# Service implementation and deployment

---



- ✧ Programming services using a standard programming language or a workflow language
- ✧ Services then have to be tested by creating input messages and checking that the output messages produced are as expected
- ✧ Deployment involves publicising the service and installing it on a web server. Current servers provide support for service installation

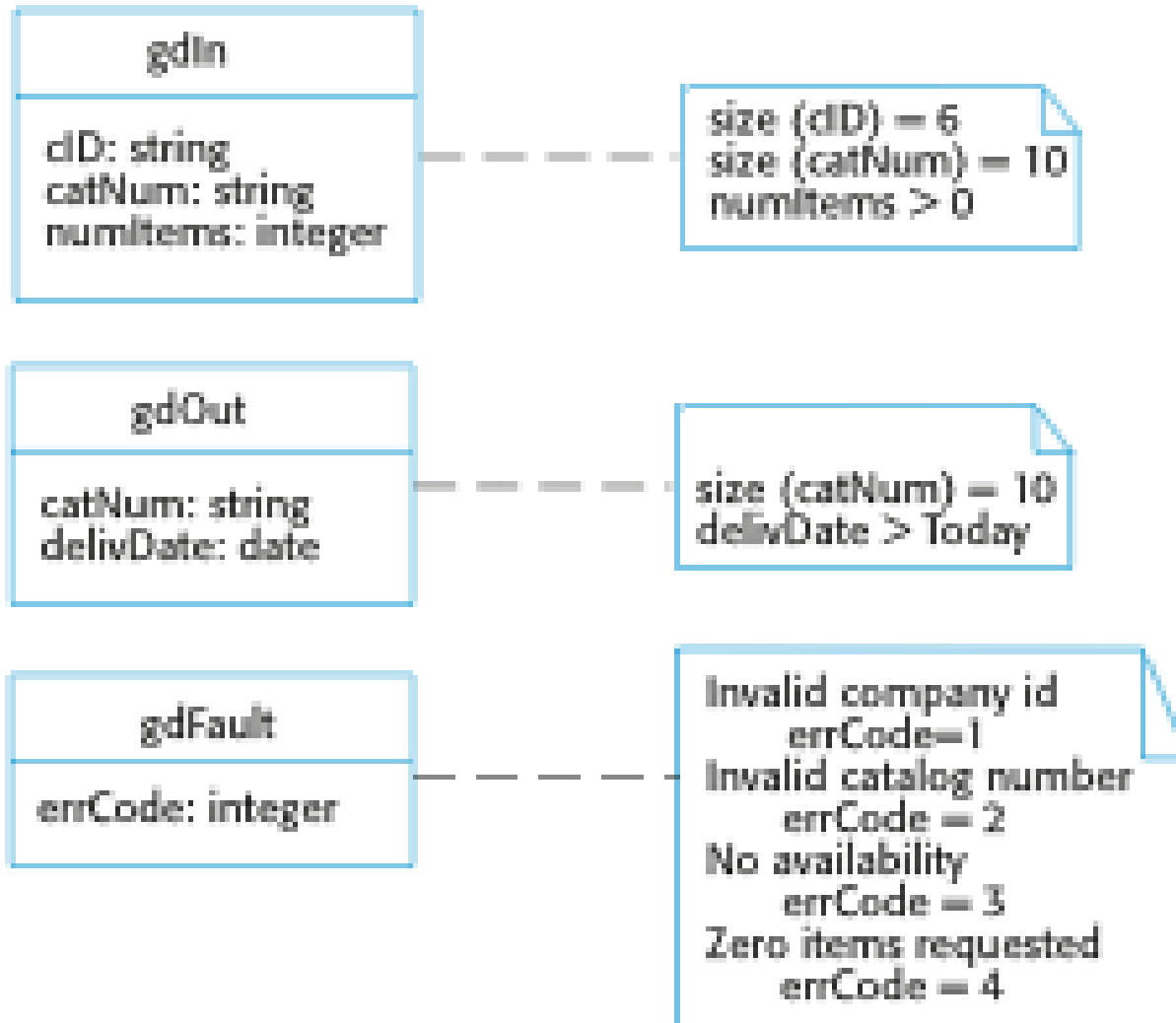


# Service descriptions

---

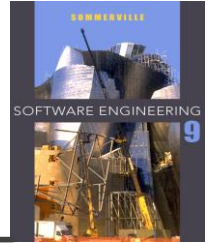
- ✧ Information about your business, contact details, etc. This is important for trust reasons. Users of a service have to be confident that it will not behave maliciously.
- ✧ An informal description of the functionality provided by the service. This helps potential users to decide if the service is what they want.
- ✧ A detailed description of the interface types and semantics.
- ✧ Subscription information that allows users to register for information about updates to the service.

# UML definition of input and output messages



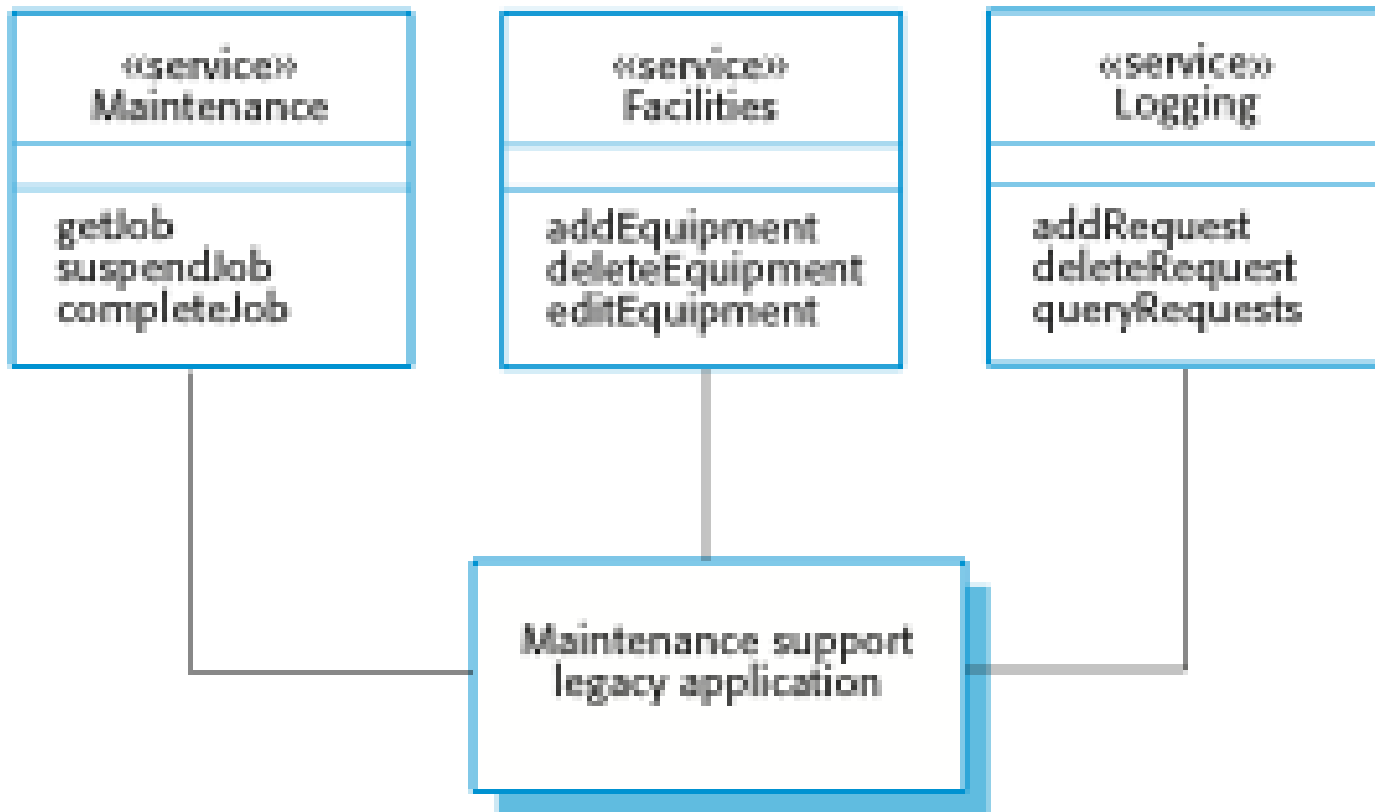
# Legacy system services

---



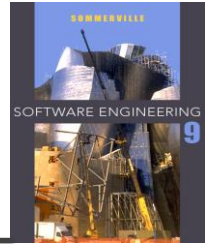
- ✧ An important application of services is to provide access to functionality embedded in legacy systems
- ✧ Legacy systems offer extensive functionality and this can reduce the cost of service implementation
- ✧ External applications can access this functionality through the service interfaces

# Services providing access to a legacy system



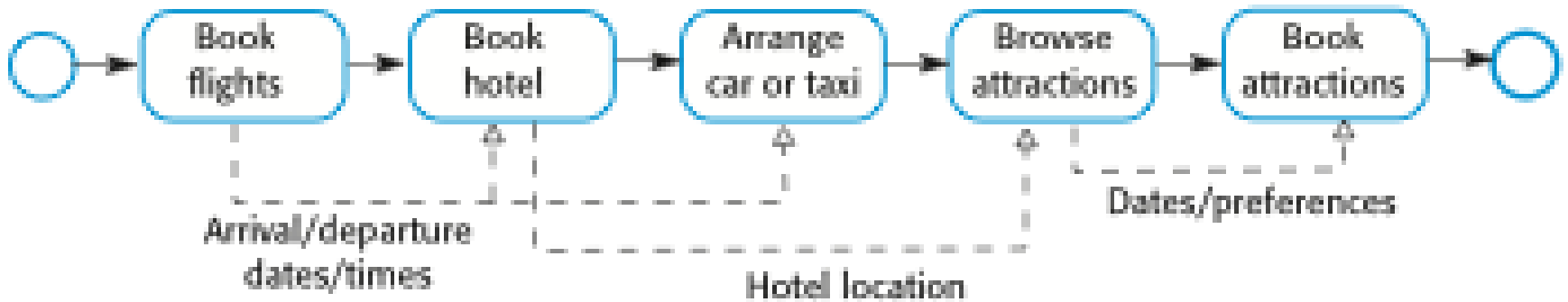
# Software development with services

---

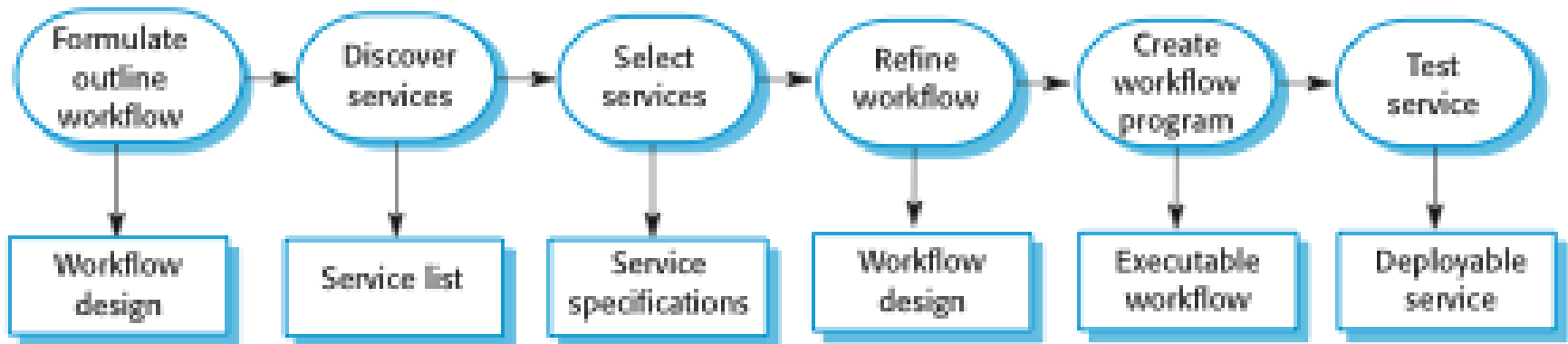
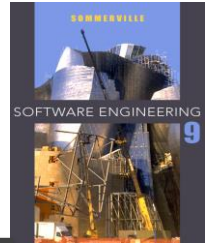


- ✧ Existing services are composed and configured to create new composite services and applications
- ✧ The basis for service composition is often a workflow
  - Workflows are logical sequences of activities that, together, model a coherent business process
  - For example, provide a travel reservation services which allows flights, car hire and hotel bookings to be coordinated

# Vacation package workflow

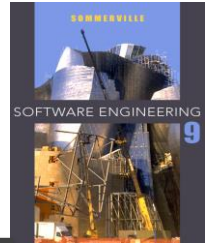


# Service construction by composition



# Construction by composition

---



## ✧ *Formulate outline workflow*

- In this initial stage of service design, you use the requirements for the composite service as a basis for creating an ‘ideal’ service design.

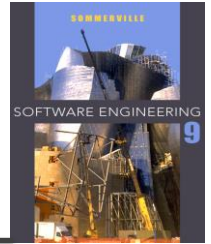
## ✧ *Discover services*

- During this stage of the process, you search service registries or catalogs to discover what services exist, who provides these services and the details of the service provision.

## ✧ *Select possible services*

- Your selection criteria will obviously include the functionality of the services offered. They may also include the cost of the services and the quality of service (responsiveness, availability, etc.) offered.

# Construction by composition



## ✧ *Refine workflow.*

- This involves adding detail to the abstract description and perhaps adding or removing workflow activities.

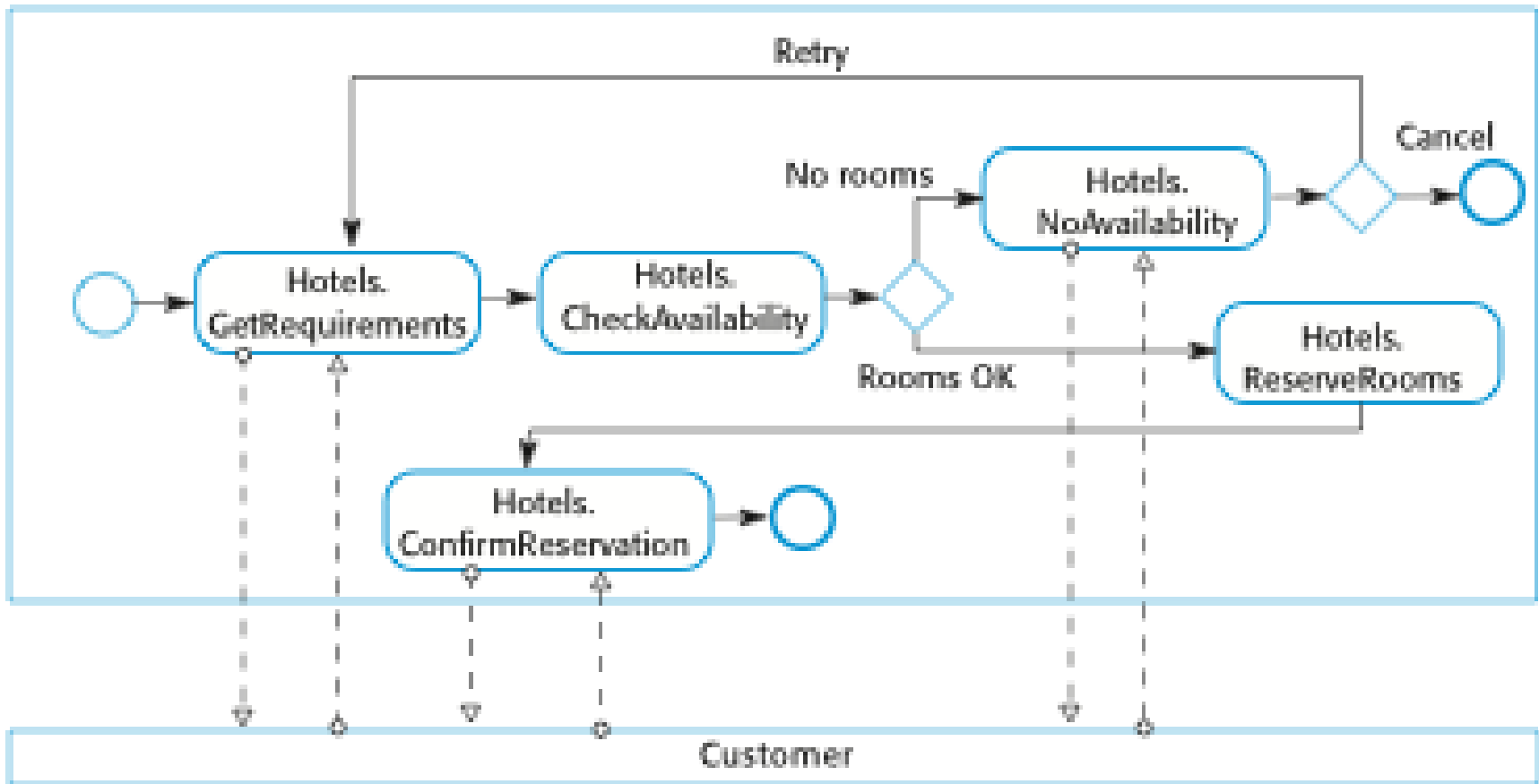
## ✧ *Create workflow program*

- During this stage, the abstract workflow design is transformed to an executable program and the service interface is defined. You can use a conventional programming language, such as Java or a workflow language, such as WS-BPEL.

## ✧ *Test completed service or application*

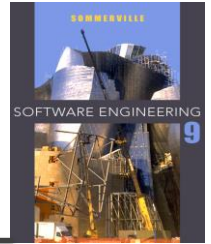
- The process of testing the completed, composite service is more complex than component testing in situations where external services are used.

# A fragment of a hotel booking workflow



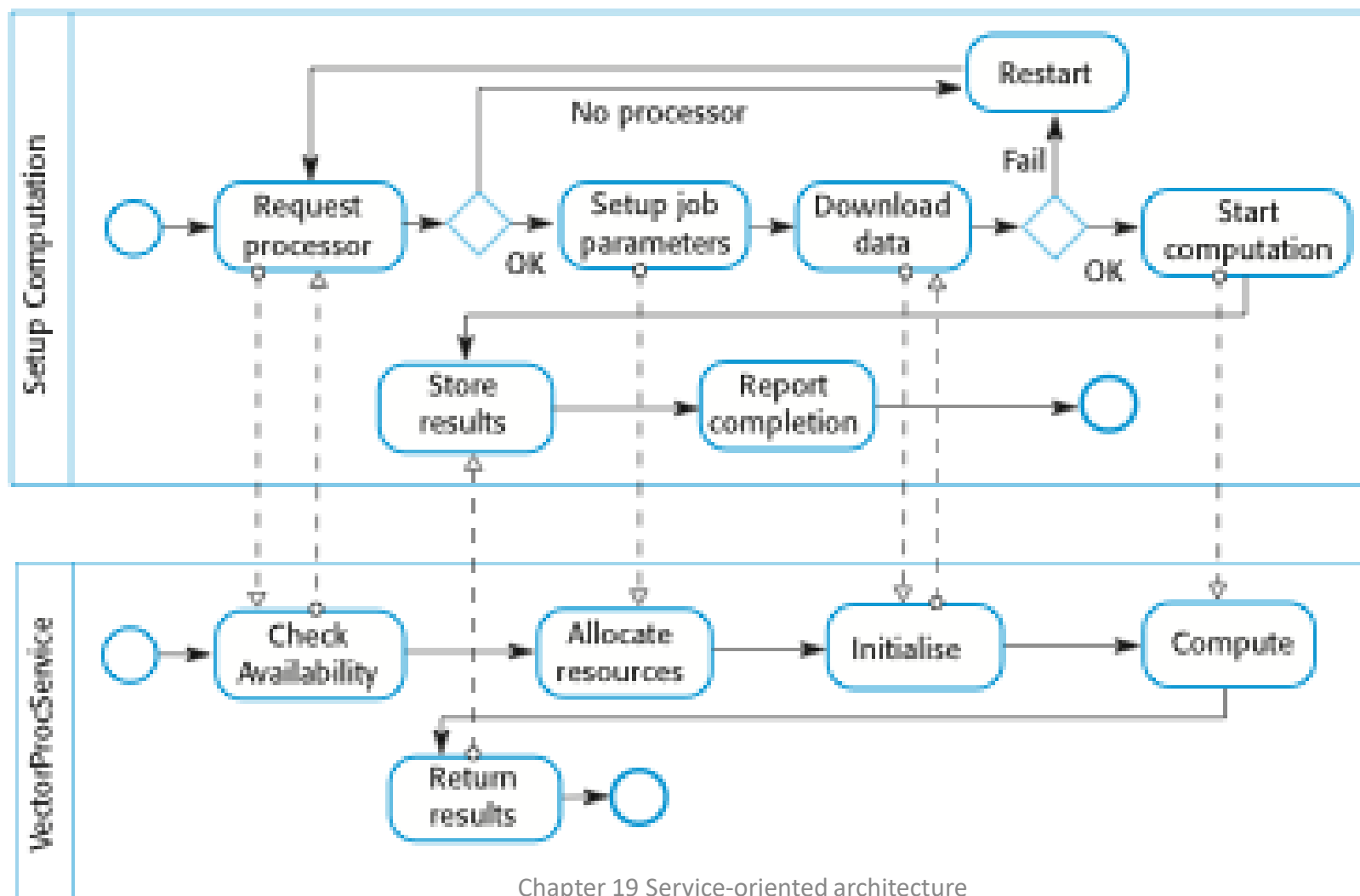
# Workflow design and implementation

---



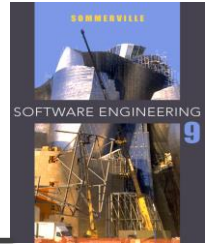
- ✧ WS-BPEL is an XML-standard for workflow specification. However, WS-BPEL descriptions are long and unreadable
- ✧ Graphical workflow notations, such as BPMN, are more readable and WS-BPEL can be generated from them
- ✧ In inter-organisational systems, separate workflows are created for each organisation and linked through message exchange

# Interacting workflows

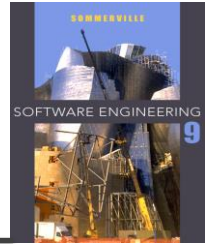


# Service testing

---



- ✧ Testing is intended to find defects and demonstrate that a system meets its functional and non-functional requirements.
- ✧ Service testing is difficult as (external) services are 'black-boxes'. Testing techniques that rely on the program source code cannot be used.



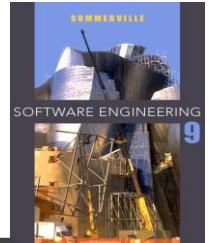
# Service testing problems

---

- ✧ External services may be modified by the service provider thus invalidating tests which have been completed.
- ✧ Dynamic binding means that the service used in an application may vary - the application tests are not, therefore, reliable.
- ✧ The non-functional behaviour of the service is unpredictable because it depends on load.
- ✧ If services have to be paid for as used, testing a service may be expensive.
- ✧ It may be difficult to invoke compensating actions in external services as these may rely on the failure of other services which cannot be simulated.

# Key points

---



- ✧ The service engineering process involves identifying candidate services for implementation, defining the service interface and implementing, testing and deploying the service.
- ✧ Service interfaces may be defined for legacy software systems which may then be reused in other applications.
- ✧ Software development using services involves creating programs by composing and configuring services to create new composite services.
- ✧ Business process models define the activities and information exchange in business processes. Activities in the business process may be implemented by services so the business process model represents a service composition.
- ✧ Techniques of software testing based on source-code analysis cannot be used in service-oriented systems that rely on externally provided services.