1. **Explain the difference between the abstract and concrete sections of the WSDL. Why is the WSDL structured this way?**

A WSDL is divided into two sections. An Abstract section and a Concrete Section.

The Abstract section is the message section which has the information like Message Type, the actual Message and the Port Type, which defines the operations that are available.

The Concrete Section has Bindings available which indicates the transport protocol that will be used when calling the operations and the Service end point.

Now, there are situations where a WSDL only has an Abstract section but no concrete section. This type of WSDL is known as an Abstract WSDL. This renders the WSDL reusable. When a WSDL also has a Concrete Section, it is bound to and defined for a specific service defined on the Concrete Section.

1. **We’ve said before that contract-first design is much better for SOA than code-first design. Given what we discussed in class, why would code-first design remain the dominant approach to the development of web service?**

In an ideal world, Contract First should be the preferred approach for designing and building Web services. But on many situations, the web service developer writes the business logic first, deploys it as web service in the web engine and lets the engine generate the WSDL for the developer who finally publishes it. This is extremely useful in teams not well versed with XML and projects can be executed by teams with junior or new members in majority. Moreover, the Code first approach is simple, less time consuming, easy and straight forward development methodology. On most cases the developers are in projects to convert legacy code to a web service. This makes it the defacto development approach.

1. **SOAP is often regarded as inefficient. Provide a concrete example of that inefficiency.**

SOAP is potentially the least efficient way for data transfer and consumption. This is primarily due to the size of SOAP file and the complex XML parsing that comes along with it. This makes it not only bad for network transport but also the complex XML parsing eats away application performance. Not to mention the learning curve involved to write efficient XML based responses. Moreover, not all platforms can parse and process SOAP responses.

In today’s, time when applications are web centric, responsive and real time there are better and simpler technologies available that alleviates most of the underlying issues with SOAP and one such technology is REST. REST offers an http transfer of data throw network pipeline in a universally accepted format called JSON. JSON is light-weight, text based structure and SON has a much smaller grammar and maps more directly onto the data structures whereas XML contains start tag and end tags. Moreover, it takes more number of characters to show data as compared to JSON. The following is an example of SOAP format and corresponding JSON.

SOAP

<?xml version="1.0"?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope" SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding" >

<SOAP-ENV:Body xmlns:m="http://www.xyz.org/quotation" >

      <m:GetQuotationResponse>

         <m:Quotation>Here is the quotation</m:Quotation>

      </m:GetQuotationResponse>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

JSON

{

"GetQuotationResponse":

{

         "Quotation": "Here is the quotation"

}

}

1. **In WSDL 2.0, the messages element was rolled into the types element. Why?**

The message element in WSDL 1.1 was mainly intended to serve as the bridge between message- and RPC-centric communication. It can be used to describe a document type message based on just one part element, or it can support RPC type (parameter-driven) messages based on multiple parts. However, its expressive power for RPC is limited. For example, it cannot describe a variable number of input parameters or a choice of responses. WSDL 2.0 addresses this industry shift by removing support for the message element altogether. It simply allows an operation to reference a type (such as an XML Schema element) directly.

1. **Provide an example of how you might use each of the four message modes we discussed in class.**
2. **In-Only**

The In-Only message mode can be used on situations requiring execution or triggering an operation that the client does not need any response back from the web service.

1. **In-Out**

The In-Out message mode can be used to execute or trigger an operation that the client expects a response back. An idea example would be to execute a function on the web service and get the output back. An example would be a Mathematical operation or getting back values from database that can be serialized to an object on the client etc.

1. **In-Optional-Out**

Like In-Out, a standard request-response message exchange pattern, except the response back to client is optional.

1. **Robust In-Only**

This is also a one-way message exchange pattern, except this is reliable. The client initiates a message and the web service responds back with a status. The response could either be a status or a fault response. If the response is a status then the transaction is complete. If the response is a fault then the client must also response back with a status.