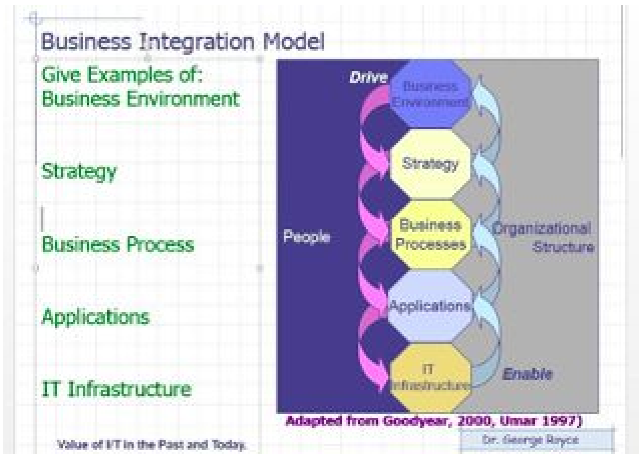
Exam 1 Review Sheet - 27 questions

Exam 1 Overview: The exam will cover material from lectures, demos, and individual and team participation activities and the readings we have covered so far. The test is closed book and closed notes.It will include multiple choice, multiple answer, short answer, and a few fill in the blank

Additional topics include:

1. Be able to describe the business integration model and give an example of how this might apply in a business.



Organizations that can keep the components of the model in alignment are most successful.

The business environment is constantly changing due to competitive threats thus we are constantly coming up with different strategy which implies changes in business processes.

The business processes are hard coded into the applications, the applications need to be able to change in conjunction with the business processes and this is determined by the IT infrastructure that is present.

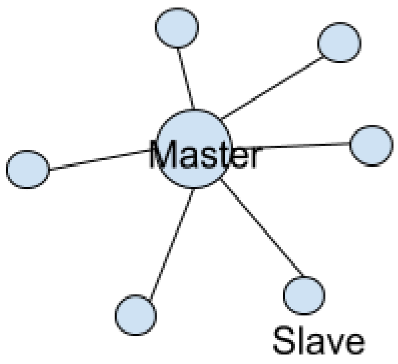
Ex: Value change management works on this model.

Business integration model drives the businesspeople to link various components (Business environment, strategy, business process, application, IT infrastructure) and enables the organization structure between them. Organizations that can keep the components of the model in alignment are most successful. Due to changes in information velocity maintaining the alignment is difficult.

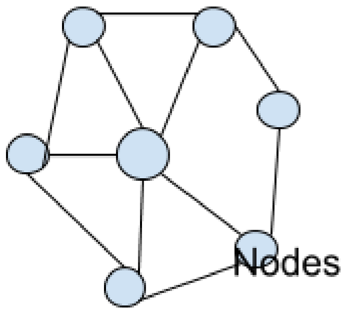
Examples:
https://accountantnextdoor.com/business-integration-models-examples-business-integration-models/#:~:text=In%20a%20nutshell%2C%20business%20integration,an%20organization%20for%20maximum%20productivity.

2. Be able to distinguish between centralized and distributed systems and homogeneous and heterogeneous systems.

<u>Centralized –</u> Processing performed in one computer or in a cluster of coupled computers in a single location. Access to the computer is via "dumb terminals"



<u>Distributed –</u> paint stores with a minicomputer in each store which runs all applications and send information back to a corporate system. Components located at networked computers communicate and coordinate their actions only by passing messages.



This definition leads to the following especially significant characteristics of distributed systems:
Concurrency of components
Lack of a global clock to help with coordination across systems
Independent failures of components – system designers must plan for this possibility

Ex: Bank and ATM Network, Massive Mulitplayer gaming, Google.

Value of Information Technology(Increased efficiency, effectiveness, business transformation) ex: **Shopping Sites, Gaming**

<u>Homogenous –</u> all servers and workstation are Microsoft Windows.
<u>Heterogeneous –</u> Application requires an IBM mainframe, UNIX servers and Microsoft Windows workstations

Example of DBMS: In a homogeneous system, all sites use the same DBMS product. In a heterogeneous system, sites may run different DBMS products, which need not be based on the same underlying data model, and so the system may be composed of relational, network, hierarchical and object-oriented DBMSs.

Homogeneous systems are much easier to design and manage. This approach provides incremental growth, making the addition of a new site to the DDBMS easy, and allows increased performance by exploiting the parallel processing capability of multiple sites.

Heterogeneous systems usually result when individual sites have implemented their own database and integration is considered at a later stage. In a heterogeneous system, translations are required to allow communication between different DBMSs. To provide DBMS transparency, users must be able to make requests in the language of the DBMS at their local site. The system then has the task of locating the data and performing any necessary translation.

3. Provide a definition of architecture, today, next minute and tomorrow architecture.
Architecture is an organized method of describing complex systems and a guide to building, acquiring and integrating information systems.

Today Architecture: As systems exist today – this minute.

Tomorrow Architecture: The target architecture. As the architecture should look in the future.

Next Minute Architecture:

- Guidelines for building, acquiring and integrating systems.
- Action-oriented – moving from today to tomorrow.
- This is where the Application and Infrastructure

  Architecture is an organized method of describing complex systems and a guide to building, acquiring and integrating information systems.

Today	Next Minute	Tomorrow
As systems exist today – this minute.	Guidelines for building, acquiring and integrating systems. Action-oriented – moving from today to tomorrow. This is where the Application and Infrastructure	The target architecture. As the architecture should look in the future.

4. Explain the process of developing and maintaining a systems architecture?
Developing and maintaining an Information Systems Architecture is a process, not just a set of documents.
The process includes:

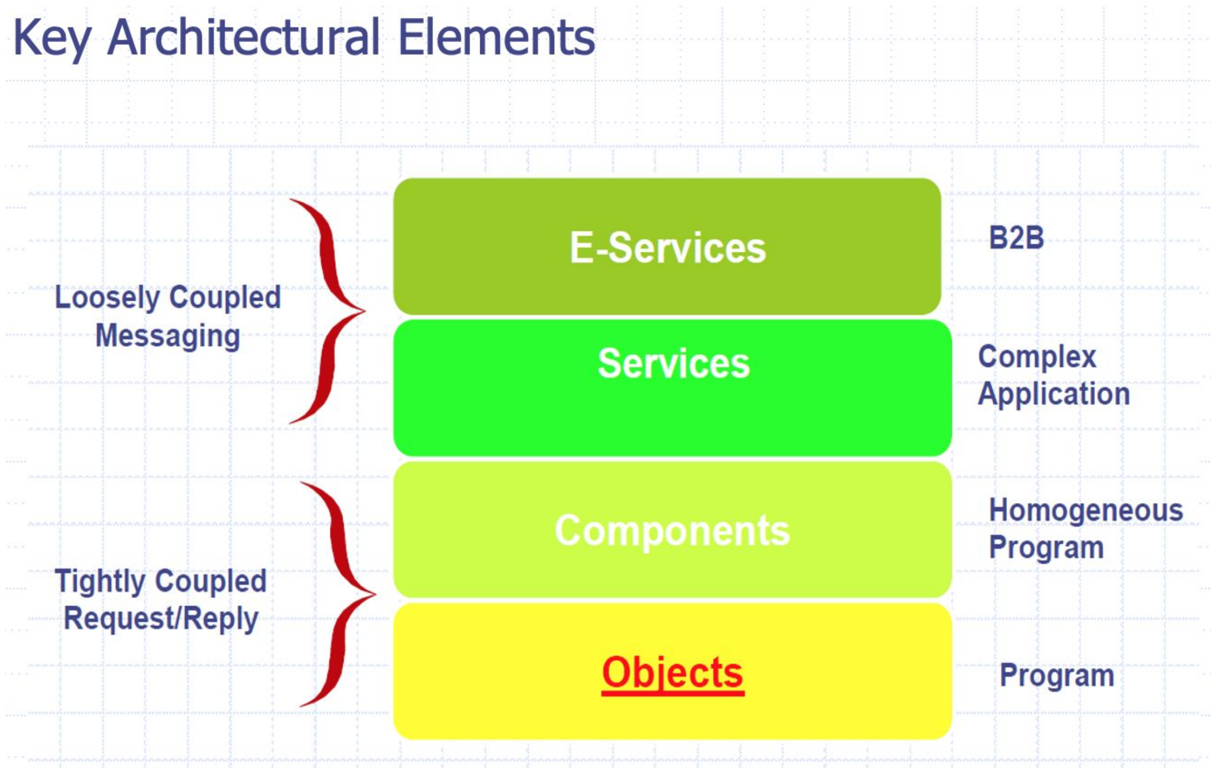
· Gathering and prioritizing business drivers - integrating architecture and strategy development with business and systems planning.
· Tracking industry trends and competitor use of technology.
· Documenting and assessing the current state of systems.
· Developing and updating the Tomorrow Architecture:

· Developing blueprints and strategies for delivering the Next Minute Architecture with a view on the Tomorrow Architecture.
· Identifying projects that deliver the Next Minute Architecture.
· Governance - Ensuring that we measure progress toward the future architecture by reviewing existing projects using the systems design and architecture review processes.

a. Gathering and prioritizing business drivers - integrating architecture and strategy development with business and systems planning.
b. Tracking industry trends and competitor use of technology.
c. Documenting and assessing the current state of systems.
d. Developing and updating the Tomorrow Architecture:
e. Developing blueprints and strategies for delivering the Next Minute Architecture with a view on the Tomorrow Architecture.
f. Identifying projects that deliver the Next Minute Architecture.
g. Governance - Ensuring that we measure progress toward the future architecture by reviewing existing projects using the systems design and architecture review processes.

5. How do system strategies relate to system architectures?

1. An Information Systems Strategy describes the approach to implementing one or more elements of the I/S Architecture.
2. A strategy considers business objectives, infrastructure and application dependencies.
3. Often, a System Roadmap or Timeline is used as an illustration of a strategy.



There is too much information about it. The detailed informations is on the slides from 12 to 22. (Class 1 -- Distributed Architecture)

An Information Systems Strategy describes the approach to implementing one or more elements of the I/S Architecture. A strategy considers business objectives, infrastructure and application dependencies. Often, a System Roadmap or Timeline is used as an illustration of a strategy.

6. Describe the major types of architecture and what they describe.

Business Process Management Architecture: is a set of activities carried out in a sequence , to realize a business objective or policy goal.

Application Software Architecture. The structure of the components/systems, their interrelationships and principles and guidelines governing their design and evolution over time.

a. Applications should be designed as a set of services communicating by sending messages to each other.
b. Services can be grouped into layers such as Presentation, Application and Data Access.
c. Application architecture focus on how should these services be distributed among physical computers to maximize performance and availability?
d. Application architecture includes 1-tier, 2 tier, 3 tier, n-tier architectures.

Data Architecture: is composed of models, policies, rules or standards that govern which data is collected, and how it is stored, arranged, integrated, and put to use in data systems and in organizations.

Infrastructure Architecture: supports the business applications.

Business Process Management (BPM) Architecture: BPM architecture is intended to ensure that a company automates its business processes. Process stages that must be performed by BPM architecture are Model processes, Model data, define formulas, define business rules, define participants, define integrations, simulate processes, run processes, and monitor processes.

Application Architecture: Application Architecture is the process of defining the framework of an organization's application solutions against business requirements. It helps to ensure that applications are scalable and reliable, manageable and assists enterprises identify gaps in functionality.

Data/Middleware Architecture: Data/Middleware architecture mediates interaction between the parts of an application, or between applications. Middleware is the software that connects software components or enterprise applications. It lies between the operating system and the applications on each side of a distributed computer network Typically, it supports complex, distributed business software applications. Middleware includes Web servers, application servers, content management systems, and similar tools that support application development and delivery.

Infrastructure Architecture: Infrastructure Architecture is a structured and modern approach for supporting an organization and facilitating innovation within an enterprise. It concerns modeling the hardware elements across an enterprise and the relationship between them.

7. Explain the pros and cons of 2 tier and 3+ tier architectures.

2-Tier Early Client Server Computing

1. Provided for printing and file services across a Local Area Network.
2. A user Interacting with a terminals attached to a mainframe.

PROS	1. Reasonably easy to set up. 2. Good Tool Support (Visual Basic .NET). 3. Low initial cost.
CONS	1. Does not scale well. •Access to “all customers” •Each client needs a separate connection to the database. •Potentially poor performance under large workloads. 2. Management of thick client software can be a problem. 3. Security can be a problem. 4. Transactional integrity can be problematic. 5. Going across a wide area network or the internet can be a real problem

3-Tier Distributed Computing

1. A third layer lies between the user and the data
2. Scalability of 3-Tier.
   - Additional application servers can be added to accommodate additional users (horizontal scaling).

PROS	1. Greater security. 2. Good performance. 3. Can support both thick and thin (browser) applications. 4. Easier to manage frequently changing software (application server should be the place for most software changes). 5. Good support for the internet. 6. Easier to integrate with legacy systems. 7. Greater reliability and flexibility.
CONS	1. Development is more complex and test environments now must have include application servers. 2. More possible points of failure than a simple 2 tier application.

Pros of 2 tier architecture:

a. Reasonably easy to set up.

b. Good Tool Support (Visual Basic .NET).

c. Low initial cost.

Cons of 2 tier architecture:

a. Does not scale well.

1. Access to “all customers”
2. Each client needs a separate connection to the database.
3. Potentially poor performance under large workloads.

b. Management of thick client software can be a problem.
c. Security can be a problem.
d. Transactional integrity can be problematic.
e. Going across a wide area network or the internet can be a real problem

Pros of 3+ tier architecture:

a. Greater security.
b. Good performance.
c. Can support both thick and thin (browser) applications.
d. Easier to manage frequently changing software (application server should be the place for most software changes).
e. Good support for the internet.
f. Easier to integrate with legacy systems.
g. Greater reliability and flexibility.

Cons of 3+ tier architecture:

a. Development is more complex and test environments now must have
b. included application servers.
c. More possible points of failure than a simple 2 tier application.

8. Describe what middleware is, the properties of middleware and what it does for applications.

Description : “Middleware is a set of common business-unaware services that enable applications and end users to interact with each other across a network. In essence, middleware is the software that resides above the network and below the business application software and denotes a reusable, expandable set of services and functions that benefit many applications in a networked environment.” (Umar, 2002)

Properties/functions:

1. Provide API to applications/ other middleware
2. Permit processes to find resources (directory service)
3. Establish connection between client/server
4. Authenticate clients & enforce security
5. Package & send requests & receive replies using an exchange protocol
6. Manage format translations between systems
7. Manage concurrent access
8. Terminate connections
9. Handle failure & synchronization, if needed.

9. Be able to explain the functions and uses of user interface (screen scraping) middleware. Give an example.(Week 2, Middleware slides)

Screen scraping is simply the process for collecting **screen display** output from one software application and translating it so that another application can **display** and **use** it.

Function: how middleware applications make new devices and applications communicate with legacy applications (i.e. PC to host integration) without changing the source code for either application

Ex: cash recycler machines can be integrated quickly with an existing teller software system without the lead-time or expense of custom development.

User Interface Strengths:

1. Can gain access to business logic only exposed through the screen interface. Newer integrated development environments are server based and generate XML messages which permit easy integration with message based systems.
2. Can be augmented with Java, Python and other contemporary languages.
3. Synchronous interaction ensures the client has most current results and always knows status of request.

User Interface Weaknesses

1. Screen scraping requires significant ongoing coordination with the team supporting the legacy systems.
2. Changes must be coordinated or the system breaks.
3. Must have the legacy system available to be useful. This may mean you run into batch window problems. Limited set of tools in this space and they will have diminishing value over time.

10. Be able to explain the functions and uses of remote procedure call middleware. Give an example.

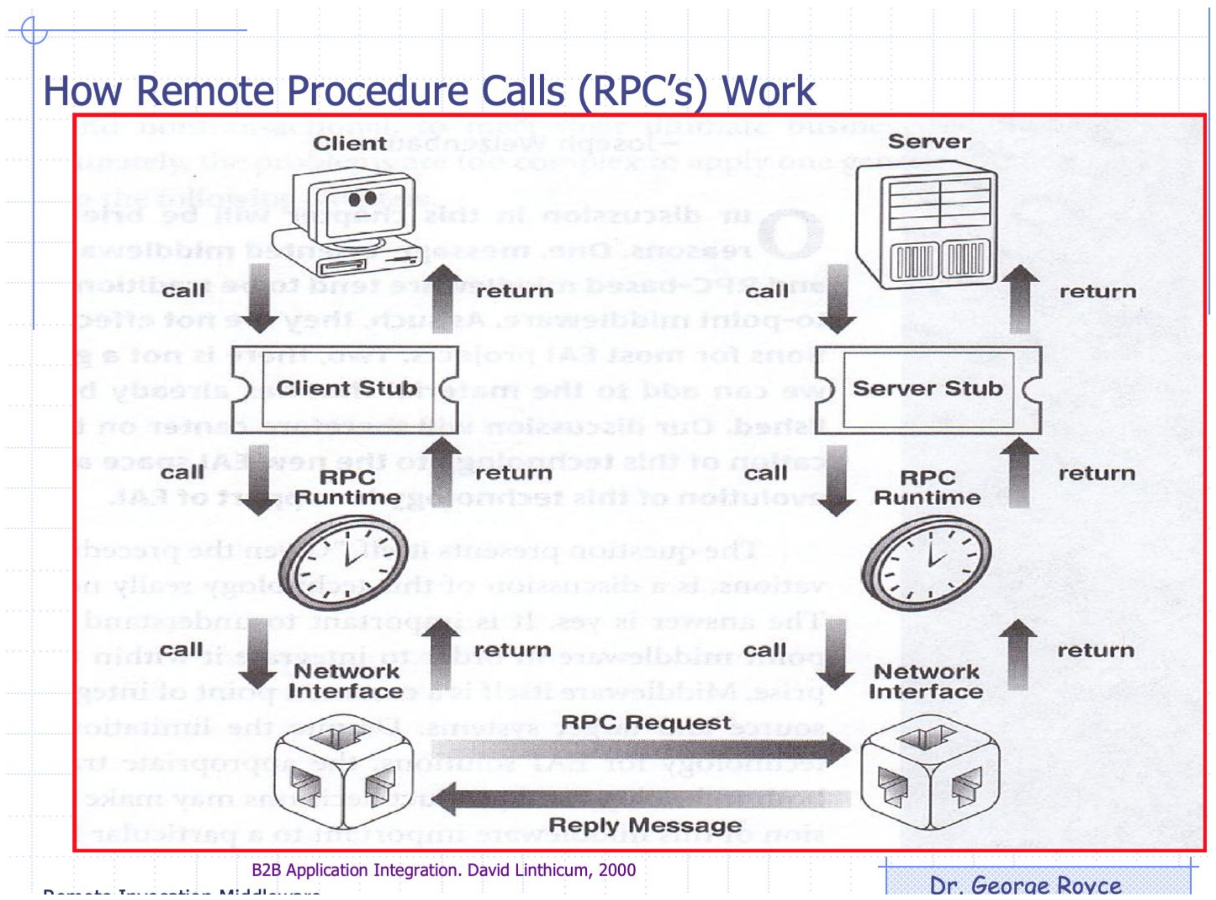
Functions:

- RPC (remote procedure call) is one of the oldest forms of middleware.
- They let a program on one computer invoke a function that can run on another computer.
- RPC is a synchronous transaction (means they stop and wait for execution on the remote platform).

Examples:

- A simple example of invoking remote procedure call (RPC) is the authentication process between a Windows client operation system and the Active Directory services (Microsoft's LDAP)
- Other examples include:
  - Open Group's DCE (Distributed Computing Open)
  - IBM's CICS (Customer Information Control System) Universal Client
  - Microsoft LDAP services uses an RPC

How Remote Procedure Calls (RPC's) Work:


How Remote Procedure Calls (RPC's) Work

B2B Application Integration. David Linthicum, 2000

Dr. George Royce

11. Be able to explain the functions and uses of message oriented middleware. Give an example.

Functions:

- Message Queuing is somewhat similar to an email system for transactions.
- Unlike most email systems, it usually features guaranteed delivery.
- Also, since the transaction is being sent as a message from one system to the other, the application does not have to stop and wait as shown below, the transactions are asynchronous.

Examples:

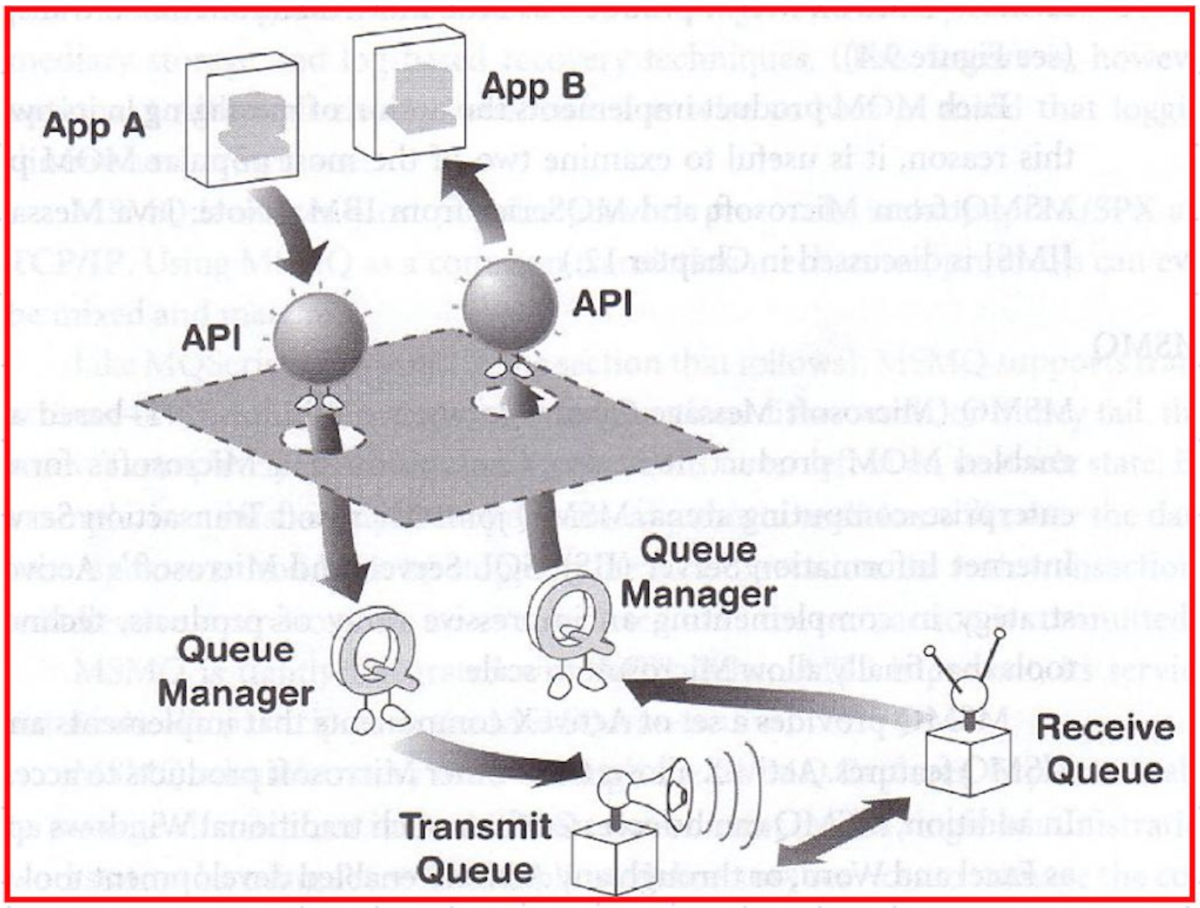
- IBM's MQ Series
- Microsoft's MSMQ
- Oracle's MQ

12. Be able to describe the concept of publish and subscribe as it relates to MOM.

Answer:

- Publish and Subscribe is also called Distributed Event-based Systems.
- Publish Subscribe middleware frees an application from the need to understand anything about the target application. All it has to do is send the information it desires to share to a location on the pub/sub broker. The broker redistributes the information to any interested applications.

How MOM (message-oriented middleware) Works?



1. Application uses standard API to send message to Queue
2. Queue Manager Routes to appropriate receiving queues.
3. Message is sent with guaranteed delivery.
4. Queue manager on receiving queue makes message available to applications.
5. Appropriate applications pick up the message and process the message.

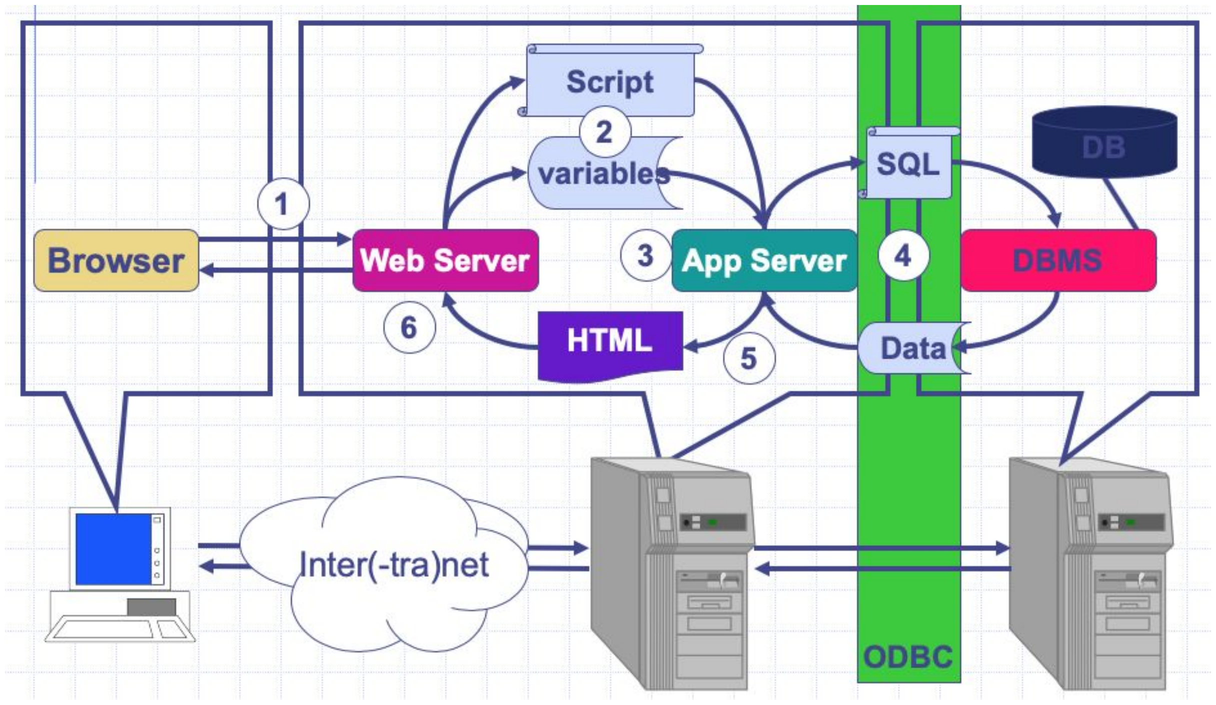
13. Be able to explain the uses of HTML, HTTP, CSS, application servers, and Python.

Answer: too much information about the use of those. I just list the definition of each which was shown on Dr.Royce's slides and how the website and application servers work.

HTML	Hypertext Markup Language - controls document structure
HTTP/HTTPS	· A protocol used to transport Web pages (which includes different sorts of objects) from one node to another · Developed at the CERN (Geneva, Switzerland) · Designed to set up & tear down connections quickly & efficiently · HTTP is a "stateless protocol" – Does not need to maintain session information with clients and server
CSS	Cascading Style Sheets – controls the look and feel

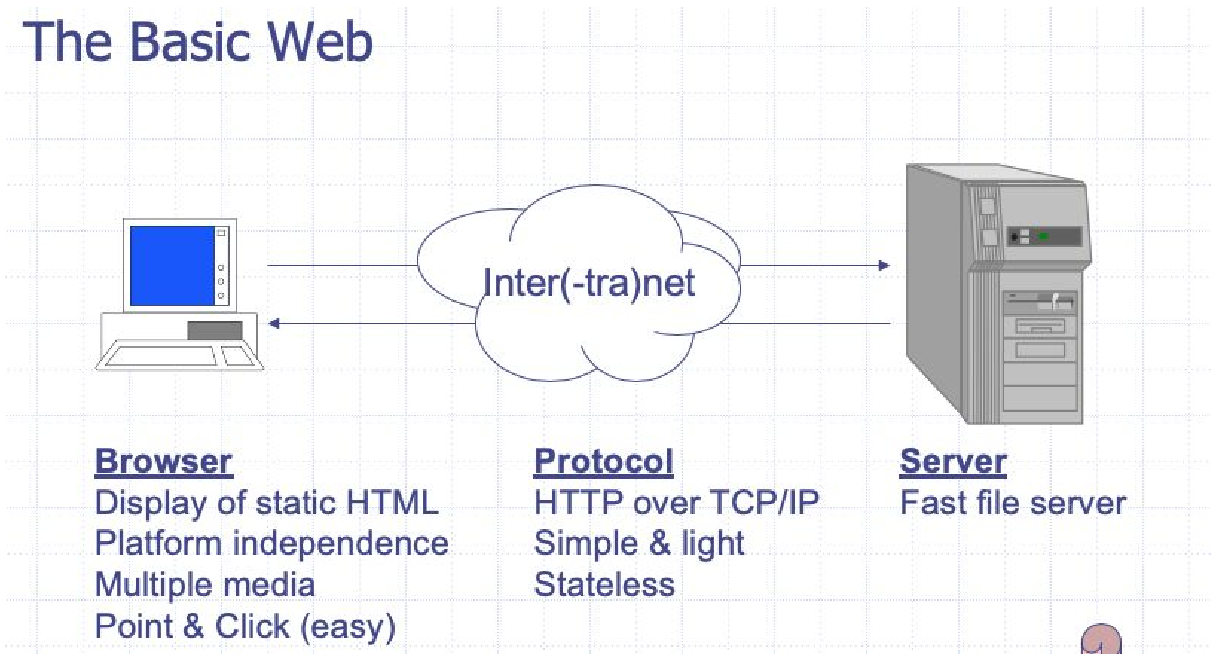
Application servers	A server (a process) providing a variety of services needed by applications, e.g. · Connectivity to the database & other information resources · Transaction management · User session management, keeping track of session state · Security functions · Load balancing, integration, fail-over · Application development environment · Our Python Django Application Server is on your machine and Heroku or PythonAnywhere
Python	Server Side Languages control process, calculation, server-side business logic

The photo below explains the whole process of web and application servers work together together (look at a detailed explain on question 14):



14. Explain the difference between the basic web and the dynamic web and what enables the dynamic web.

The basic web:



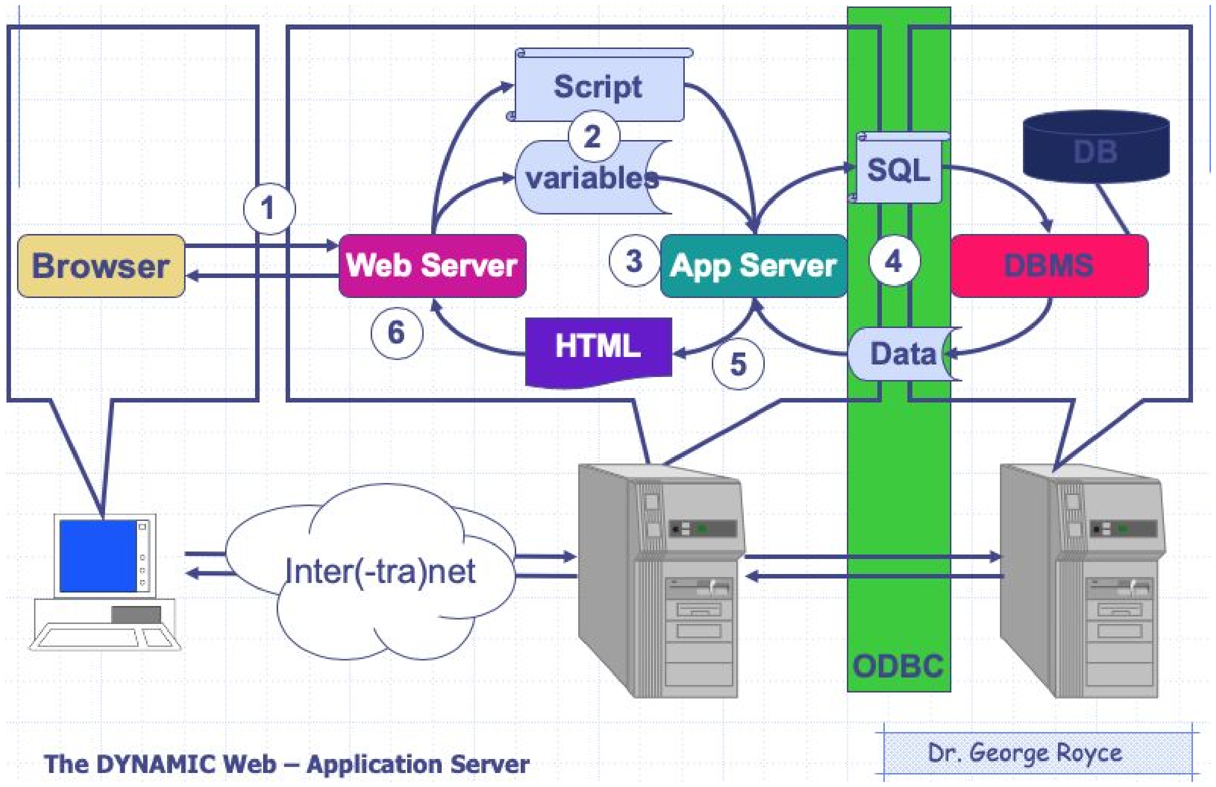
The Basic Web

Browser
Display of static HTML
Platform independence
Multiple media
Point & Click (easy)

Protocol
HTTP over TCP/IP
Simple & light
Stateless

Server
Fast file server

The dynamic web:



This diagram provides both a physical view (the hardware and networks) and a logical view (which processes run where).

1. Browser requests a page, which may have a special extension like .cfm or .pl to indicate that it is really a program rather than a page. If User is looking at a form, has provided some input, and clicked on 'submit', then the request will include variables in the form of attribute/value pairs
2. If the request is for a static page, the Web Server finds the page (images, etc.) and returns them as in the Basic Web. If the request references a script, the Web Server sends the Script along with any Variables to the Application Server.
3. The Application Server processes {interprets, invokes} the script.

4. Some scripts will request connections to the database and provide queries which are to be executed. The Application Server establishes the connection, sends the query, receives the results.
5. With the data results and the HTML skeleton provided by the script, the Application Server creates an HTML Page
6. The HTML page is sent back to the Web Server for delivery back to the browser

15. What services does an application server provide? Name at least three different application servers. Be able to explain how several application servers can handle a large application with thousands of users. Describe what happens to the customer transactions when one of the application servers is no longer available.

Answer:

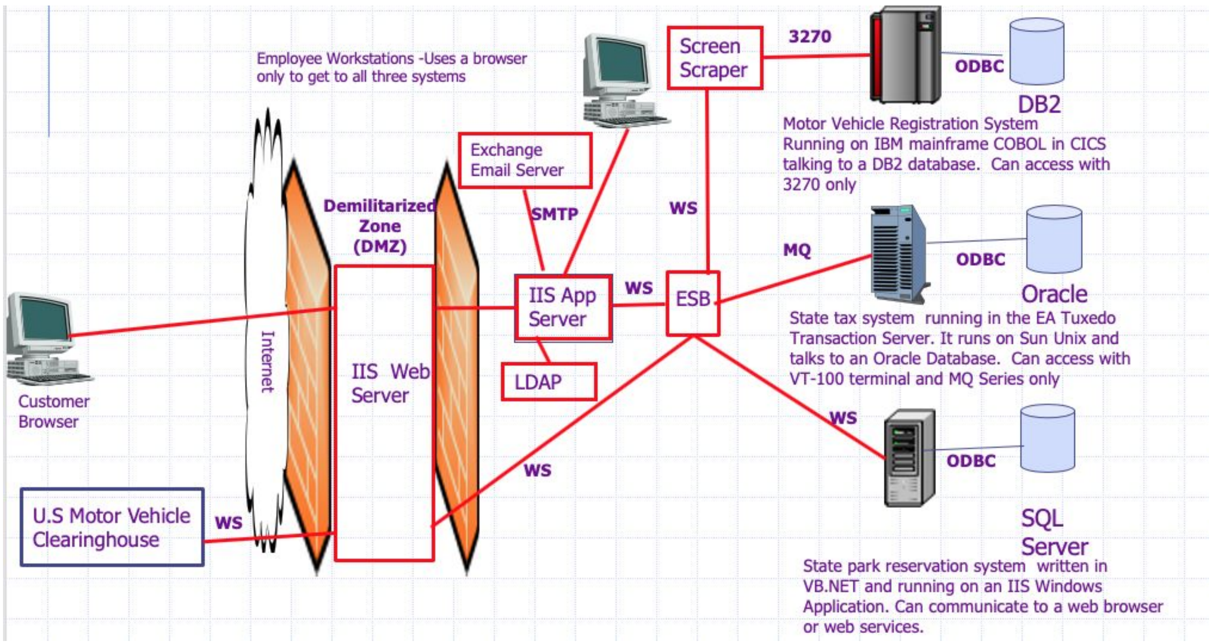
A server (a process) providing a variety of services needed by applications, e.g.

- Connectivity to the database & other information resources
- Transaction management
- User session management, keeping track of session state
- Security functions
- Load balancing, integration, fail-over
- Application development environment

Name at least three different application servers:

- JavaScript: Wakanda Serve, Node.js
- Python: Gunicorn, mod_wsgi, Tornado, Google App Engine

There is an example from Dr. Royce's slide about how several application servers work together.


Employee Workstations -Uses a browser only to get to all three systems
Screen Scraper
3270
ODBC
DB2
Motor Vehicle Registration System Running on IBM mainframe COBOL in CICS talking to a DB2 database. Can access with 3270 only
Exchange Email Server
SMTP
WS
Demilitarized Zone (DMZ)
MQ
ODBC
Oracle
IIS App Server
WS
ESB
State tax system running in the EA Tuxedo Transaction Server. It runs on Sun Unix and talks to an Oracle Database. Can access with VT-100 terminal and MQ Series only
Internet
IIS Web Server
LDAP
Customer Browser
WS
ODBC
WS
SQL Server
U.S Motor Vehicle Clearinghouse
WS
State park reservation system written in VB.NET and running on an IIS Windows Application. Can communicate to a web browser or web services.

When one of the application servers is no longer available, the customer transactions will stop working, because when customer login the website, the error will happen like this "The resource you requested is no longer available from the servers or the specified user no longer has access to"(from https://discussions.citrix.com/topic/257751-the-resource-you-requested-is-no-longer-available-from-the-servers-or-the-specified-user-no-longer-has-access-to-it/)

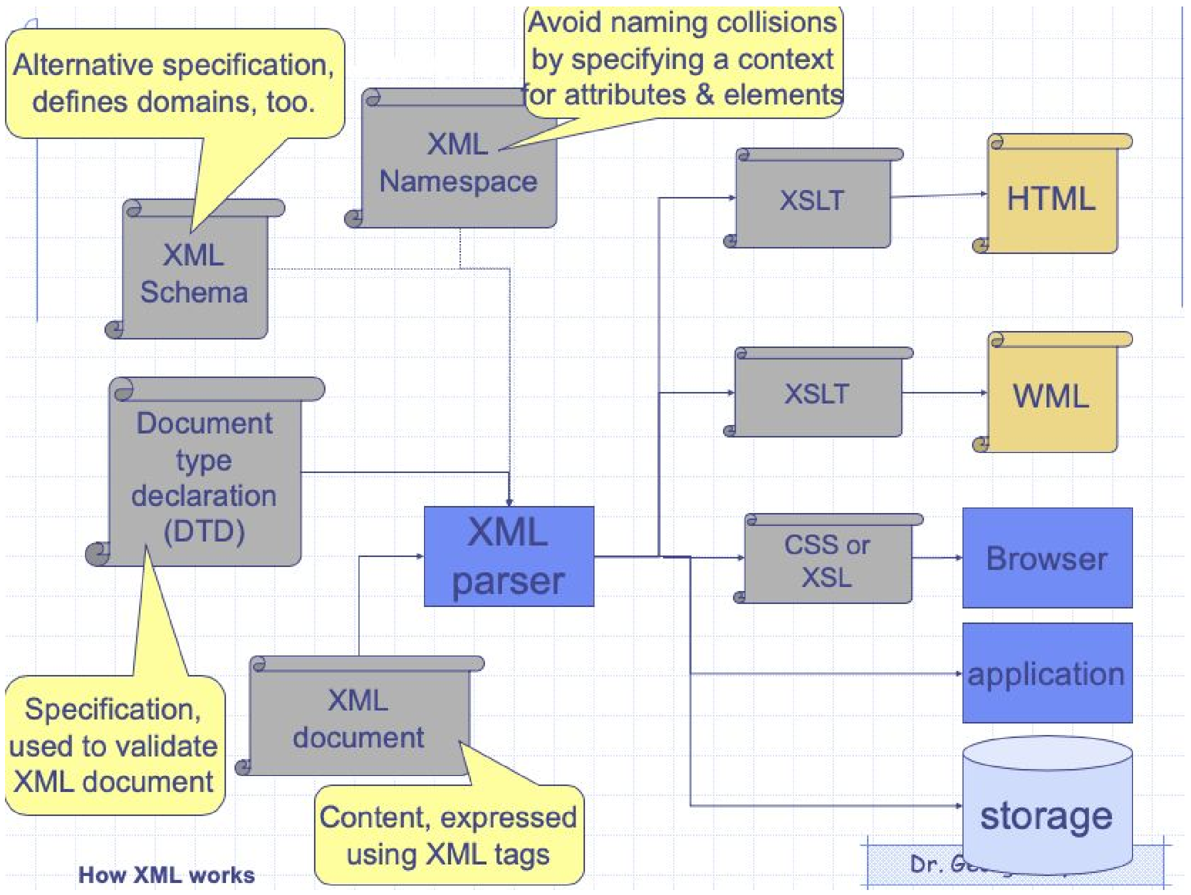
16. What is XML and a DTD and how is it used in application development and integration?

XML: **Extensible Markup Language**. It is an open messaging and document standard providing a cross-platform, portable mechanism for exchanging data and documents.

- XML is based on a tagged document or message format.
- The tag language has been developed from older markup standards such as SGML.
- The XML standard was first created by the World Wide Web Consortium (W3C) as a means of exchanging data and documents over the Internet.

DTD: **Document type declaration**

How it is used:


Alternative specification, defines domains, too.
Avoid naming collisions by specifying a context for attributes & elements
XML Namespace
XML Schema
XSLT
HTML
XSLT
WML
Document type declaration (DTD)
XML parser
CSS or XSL
Browser
application
storage
Specification, used to validate XML document
XML document
Content, expressed using XML tags
How XML works
Dr. G

In the XML world, the content (XML document) is separated from the definition of the structure of that document (DTD or XML Schema), which is separated from the specification of how to visually represent that content (Extensible Style Language Transformation (XSLT)).

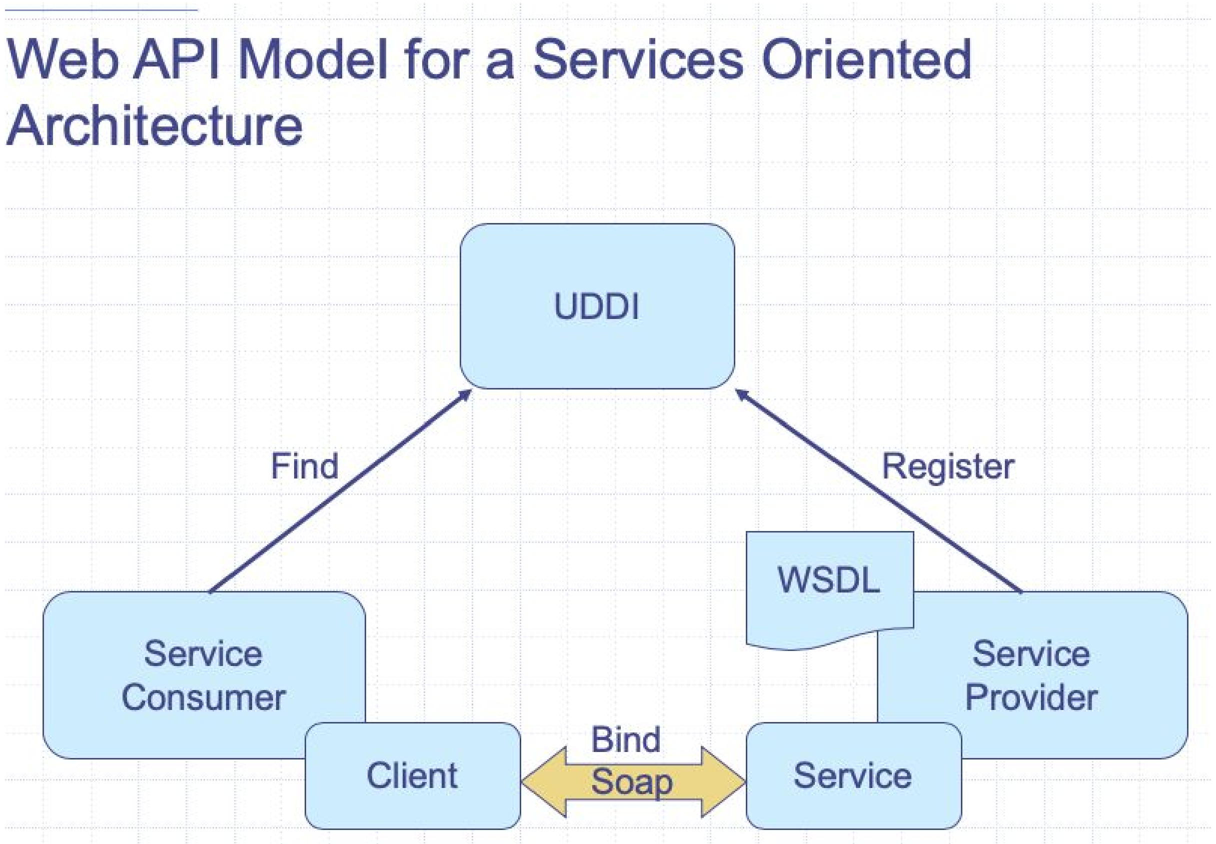
An XML document can be used in a variety of ways. First, it can represent content that is to be displayed. Here, we have a number of alternatives: It can be displayed directly use a Cascading Style Sheet (CSS) or Extensible Stylesheet Language (XSL) stylesheet to indicate how to display the various elements of the XML document. Second, the XML can be converted into another language, like HTML or WML, via the Extensible Style Language Transformation (XSLT). Third, it can contain content that will be used by another application. This would be a prime use in B2B applications. Fourth, it would be content that is to be stored within a database.

17. What is voiceXML?

Answer: Voice XML - An Emerging Standard --General Magic has used Voice XML for navigation of a web site using voice and speech recognition. General Motors ONStar is and example of this technology.

Voice XML - An XML Language Standard ---- VRU/IVR and Web applications can now be developed by developers trained in Java and XML.

18. Be able to define the essential parts of a web service and draw a simple diagram illustrating how a web service operates. The parts include: SOAP, WSDL, UDDI, Service Consumer (client) and Service Consumer (server).

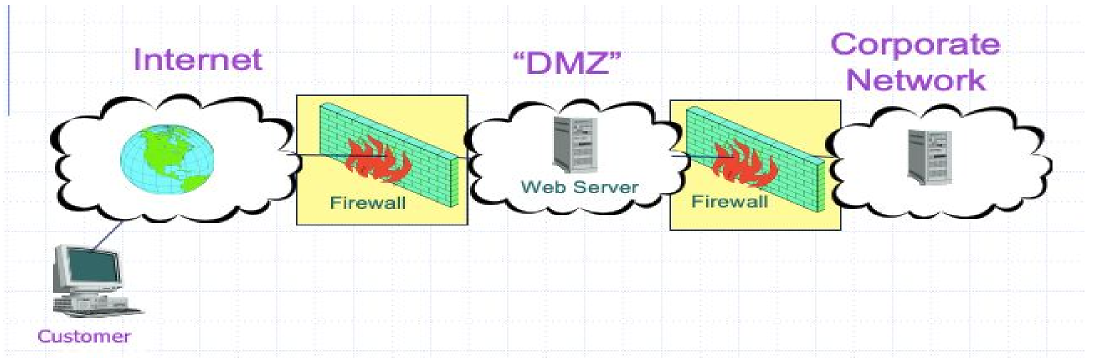


Web API Model for a Services Oriented Architecture

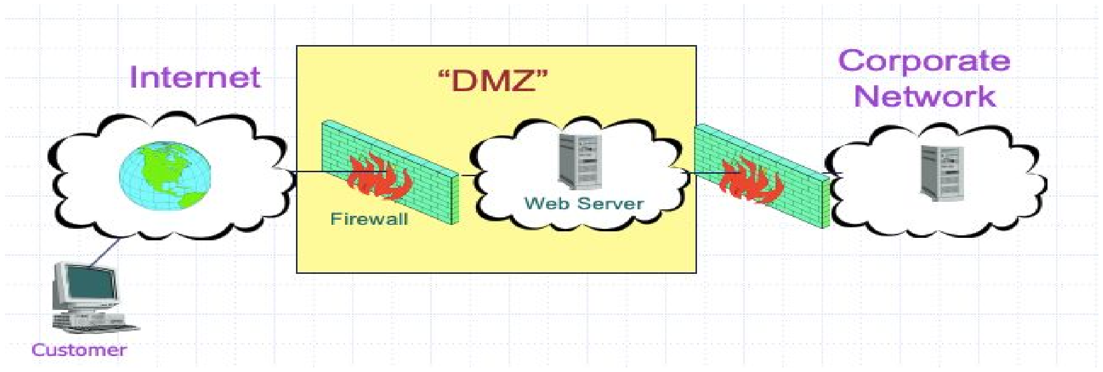
SOAP: Simple Object Access Protocol, an XML-based messaging protocol for invoking program logic over HTTP
WSDL: Web Services Description Language, an abstract way to describe the capabilities of Web services
UDDI: Universal Description, Discovery, and Integration, a specification for an Internet-wide registry of web services and their metadata.

19. Explain the following terms and be able to diagram a scenario using these terms: firewall, DMZ, hardened operating system, SSL, intrusion detection, redirector, LDAP, digital certificate. Be able to use these in a systems integration problem described above.

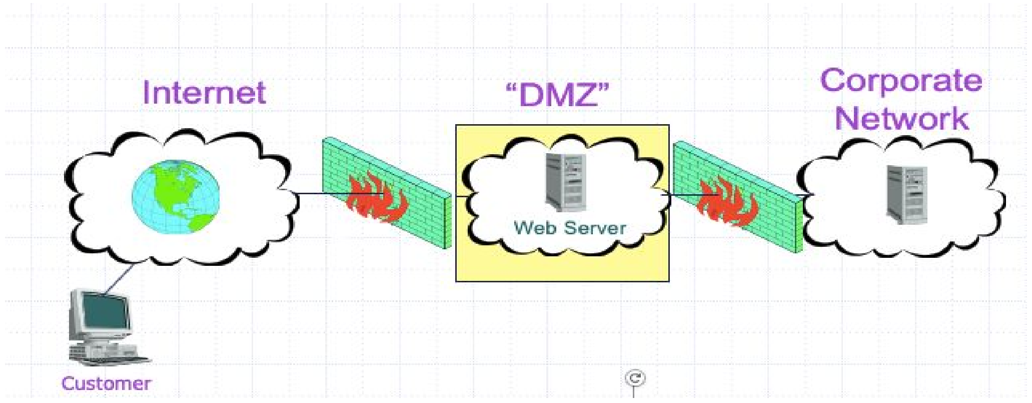
Firewall - A firewall is an access control system. By using firewall policies, access to the network can be restricted.



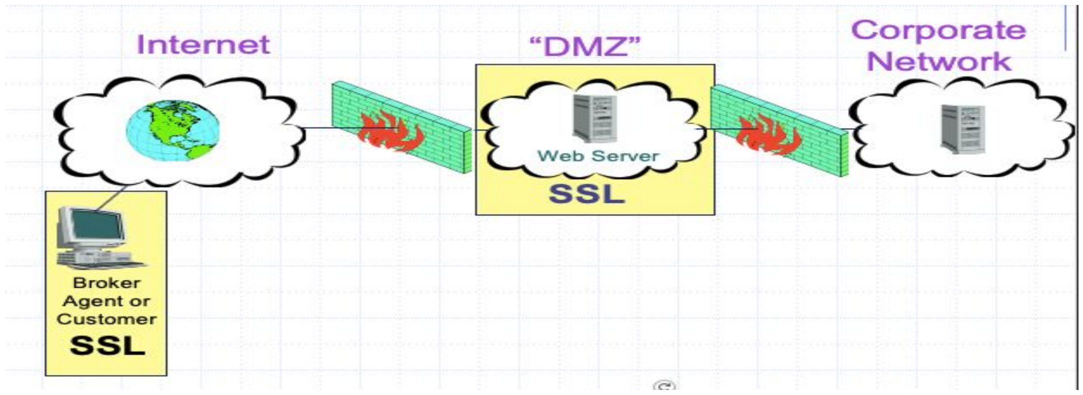
DMZ - Demilitarized zone
It is a monitored network for our customers and business partners to request services from systems on our corporate network, without entering the corporate network.



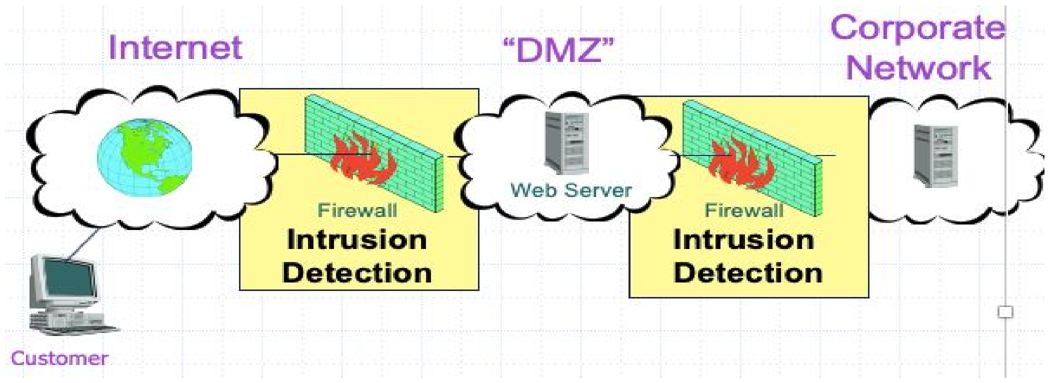
Hardened operating system - It is a version of an operating system which has only a small set of secured services to limit the possibility of hacking into the system.



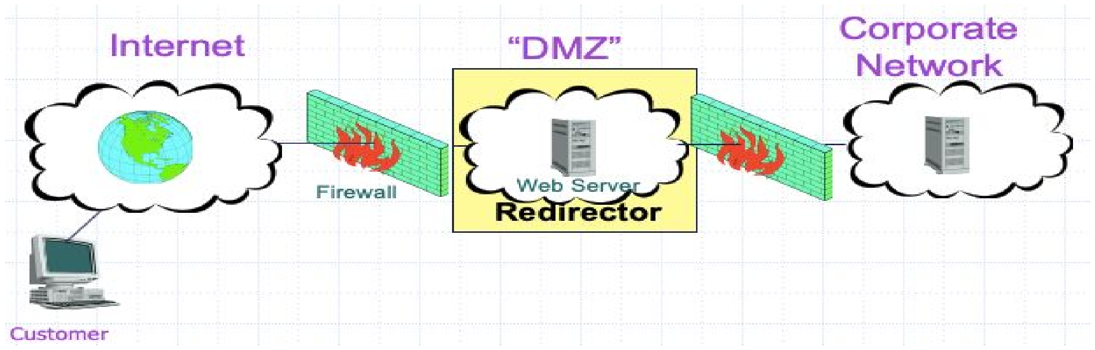
Secure socket layer(SSL) - It is a software that provides encryption of web server transactions.



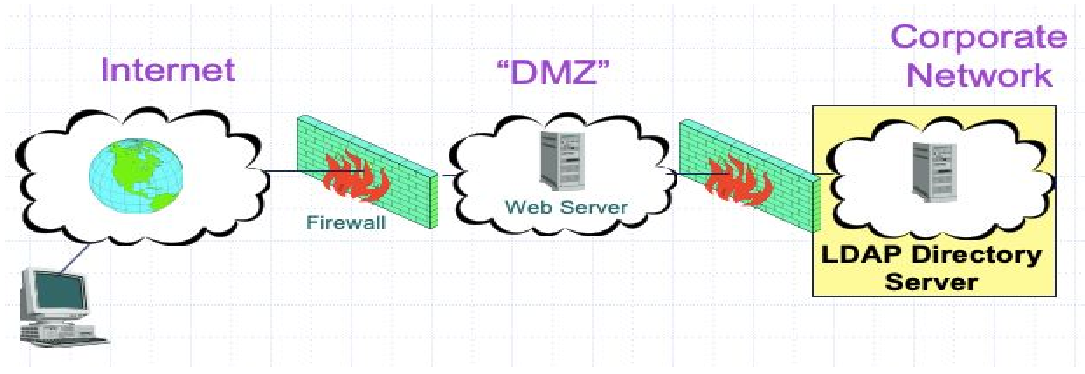
Intrusion detection - It monitors network traffic to detect and respond to suspicious activity before systems are compromised.



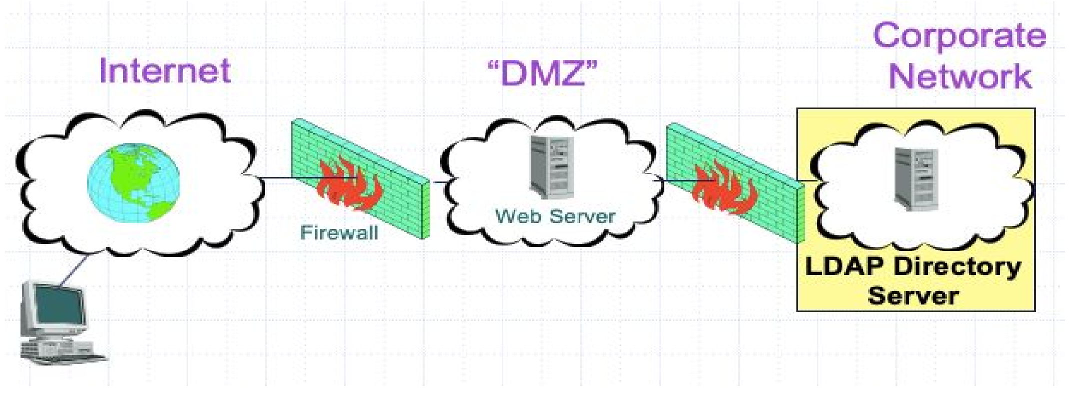
Redirector - software that accepts requests for information from a customer or supplier and sends the request through a firewall to retrieve the correct information from a system on the corporate network.



LDAP - An LDAP directory server is used to authenticate the customer or business partner and authorize his/her use of services.



Digital certificates - A certificate is a digital identification that contains information about the owner, including the public/private key that is used for encryption and digital signatures. Certificates can be issued for a person and a server. They are stored in a directory.



20. Describe the difference between traditional computing and cloud computing in terms of acquisition, business, access, and technical models.

Models	**Traditional computing**	**Cloud Computing**
Acquisition	Buy assets and build technical architecture	Buy service.
Business	Pay for fixed assets and administrative overhead	Pay based on use.
Access	Internal network or intranet, corporate client	Internet, any device.
Technical models	Single tenant, nonshared, static	Scalable, elastic, dynamic, multi-tenant

21. Cloud computing is still evolving. Explain the similarities and differences between the offerings provided by these three big players we talked about in class: Salesforce/Force, Amazon and Google's App Engine for Python and Java.

Similarities

- Vendors host the data and the app logic
- Enterprises have the flexibility to acquire new resources for computing and storage
- Little IT knowledge needed to get started with a complex solution

Difference:

- SalesForce focused on business professional for creating CRM solutions that will talk easily with other systems.
- Amazon sells the idea of doing the entire IT function in the cloud. No worries about scalability
- Google App Engine - is python and java playground and currently limited to google apps users.

22. Describe the development, code promotion and integration testing processes we are using which simulates processes used by many businesses. What additional steps are required to promote code into a production instance and ensure that there are no defects in the application?

Three environments are needed.

1. **Development** - used for coders to build the app. At the end of each iteration/sprint the updates are pushed to Testing.
2. **Testing** - In this environment testers play with the software. Bugs are recorded, prioritised and tracked. Coders do the changes in the development environment
3. P**roduction** - A tested version is pushed here to the end user according to the work graphic. This what the user get to use everyday.

23. Explain how a MVC framework application is built and the role of the database, the model, view and controller. Compare using a MVC framework to creating an application in native code only. What are the advantages and limitations? Explain the difference between an MVC and an MVT framework such as Django?

What is MVC?

Architectural design pattern which works to separate data and UI for a more cohesive and modularized system

Model represents the data model

> “Manages behavior and data of the application domain”

View represents the screen(s) shown to the user

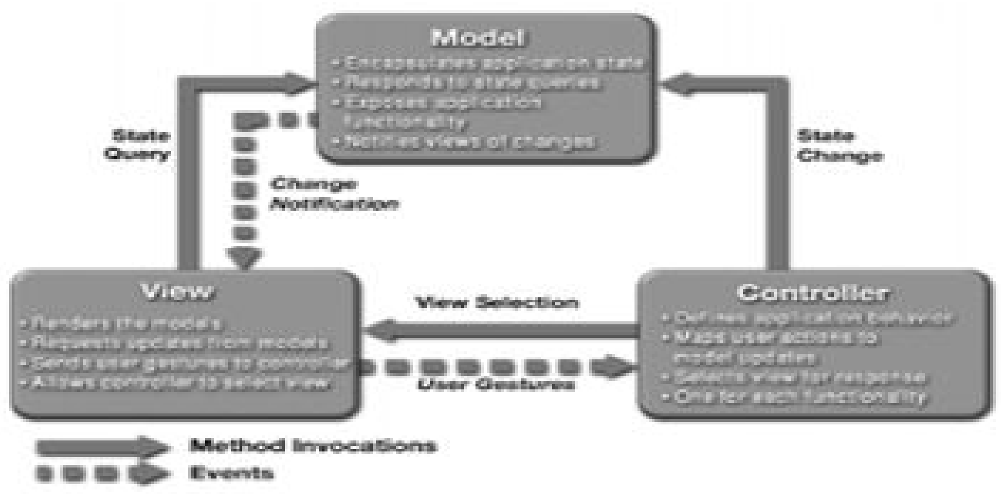
> “Manages the graphical and/or textual output to the portion of the bitmapped display that is allocated to its application”

Controller represents interactions from the user that changes the data and the view

> “Interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate” (Burbeck)

How Does it Work?

- User inputs a command e.g., HTTP GET, POST usually from a web page.
- Controller handles input and updates model (which updates the database) or changes the view
- View, which relies on model to show data to user, updates if necessary, on the next request
- Rinse and Repeat


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
State Change
Change Notification
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Why Use M-V-C?

- Provides a logical structure for heavily interactive system
- Adheres to good engineering design principles and practices
- Information hiding, less tight coupling, simplicity, etc.
- Delegated control style

MVC is the separation of **m*odel, *v*iew and *c*ontroller — nothing more, nothing less. It's simply a paradigm; an ideal that you should have in the back of your mind when designing classes. Avoid mixing code from the three categories into one class.

For example, while a table grid *view* should obviously present data once shown, it should not have code on where to retrieve the data from, or what its native structure (the *model*) is like. Likewise, while it may have a function to sum up a column, the actual summing is supposed to happen in the *controller*.

A 'save file' dialog (*view*) ultimately passes the path, once picked by the user, on to the *controller*, which then asks the *model* for the data, and does the actual saving.

This separation of responsibilities allows flexibility down the road. For example, because the view doesn't care about the underlying model, supporting multiple file formats is easier: just add a model subclass for each.

The MVC pattern has been heralded by many developers as a useful pattern for the reuse of object code and a pattern that allows them to significantly reduce the time it takes to develop applications with user interfaces.

The model-view-controller pattern proposes three main components or objects to be used in software development:

- A *Model* , which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.
- A *View* , which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)
- A *Controller* , which represents the classes connecting the model and the view,

Django(MVT Framework):

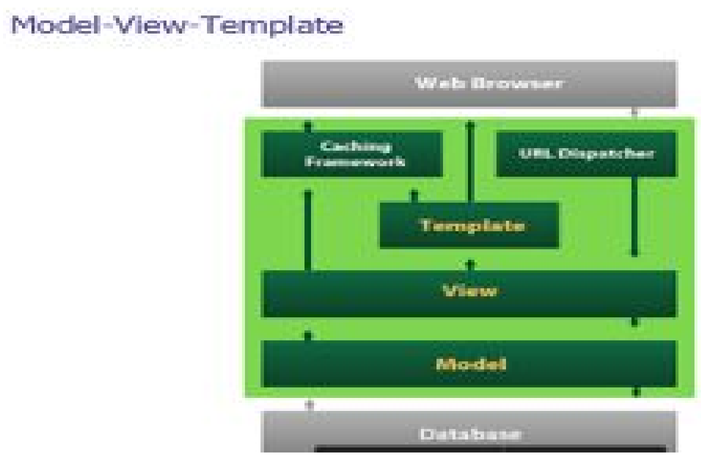
Features such as CRUD (create, read, update and delete) are security authentication and authorization are built into the framework.

It uses Model and View and is best to drive out the data model first which can is used by Django

How does Django Work?

Here’s what happens when a visitor lands on your Django page:

- First, Django consults the various URL patterns you’ve created and uses the information to retrieve a view.
- The view then processes the request, querying your database if necessary.
- The view passes the requested information on to your template.
- The template then renders the data in a layout you’ve created and displays the page.



Major Features of Django

- Complete Object Relational Mapper (ORM)
- Built in form serialization and validation system
- A serialization system that can produce and read XML and/or JSON representations of Django model instances
- An interface to Python's built-in unit test framework
- A built in app server/web browser
- An extensible authentication system
- A dynamic and easy to use administrative interface.

MVC uses logical structure for heavily interactive systems unlike web applications without a framework.
MVC adheres to good engineering design principles and practices like information hiding, less tight coupling, simplicity, delegated control style.

24. Be able to answer basic questions about python logic as described in the python overview.

Python is a general purpose programming language. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly braces or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java.
Python is used in many areas including:

- Web development
- Bioinformatics
- Data sciences, Data Mining, etc.

Some basic Python Coding Rules

- Python relies on proper indentation. Incorrect indentation causes an error.
- The standard indentation is four spaces.
- With implicit continuation, you can divide statements after parentheses, brackets, and braces, and before or after operators like plus or minus signs.
- With explicit continuation, you can use the \ character to divide statements anywhere in a line.

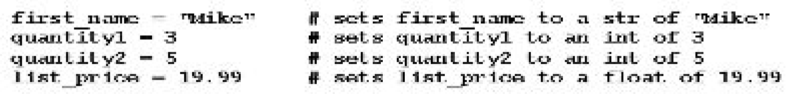
Syntax for calling any function

Function_name([arguments])
Ex: print([data])

Three Python data types

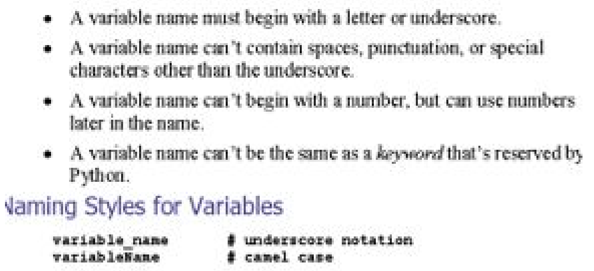
- Int
- str
- float

Initializing Variables and Assigning Data



```
first_name = "Mike"     # sets first_name to a str of "Mike"
quantity1 = 3           # sets quantity1 to an int of 3
quantity2 = 5           # sets quantity2 to an int of 5
list_price = 19.99      # sets list_price to a float of 19.99
```

Naming variables:

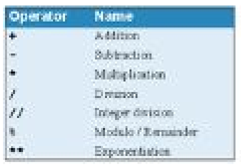


- A variable name must begin with a letter or underscore.
- A variable name can't contain spaces, punctuation, or special characters other than the underscore.
- A variable name can't begin with a number, but can use numbers later in the name.
- A variable name can't be the same as a *keyword* that's reserved by Python.

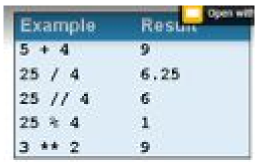
Naming Styles for Variables

```
variable_name      # underscore notation
variableName       # camel case
```

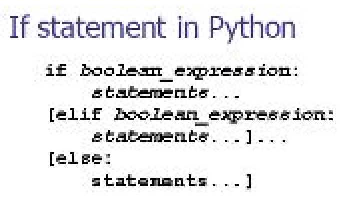
Arithmetic Operations:



Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Integer division
%	Modulo / Remainder
**	Exponentiation

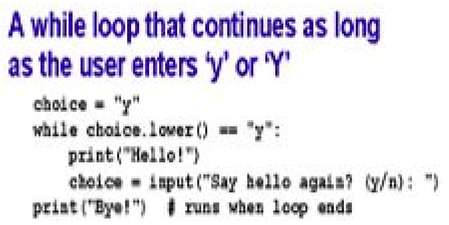


Example	Resu
5 + 4	9
25 / 4	6.25
25 // 4	6
25 % 4	1
3 ** 2	9



If statement in Python

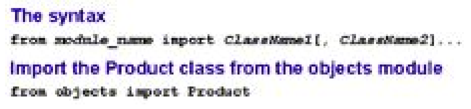
```
if boolean_expression:
    statements...
[elif boolean_expression:
    statements...]...
[else:
    statements...]
```



A while loop that continues as long as the user enters 'y' or 'Y'

```
choice = "y"
while choice.lower() == "y":
    print("Hello!")
    choice = input("Say hello again? (y/n): ")
print("Bye!")  # runs when loop ends
```

Import a class:

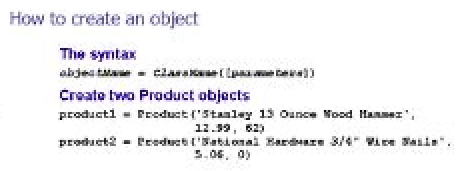


The syntax

```
from module_name import ClassName1[, ClassName2]...
```

Import the Product class from the objects module

```
from objects import Product
```



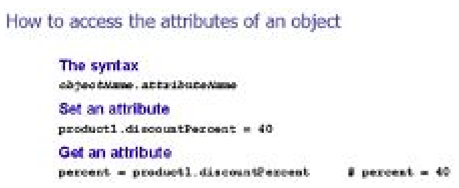
How to create an object

The syntax

```
objectName = ClassName([parameters])
```

Create two Product objects

```
product1 = Product('Stanley 13 Ounce Wood Hammer',
                   12.99, 62)
product2 = Product('National Hardware 3/4" Wire Nails',
                   5.06, 0)
```



How to access the attributes of an object

The syntax

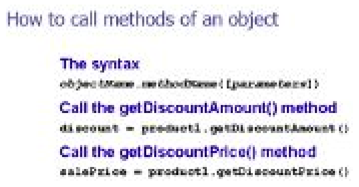
```
objectName.attributeName
```

Set an attribute

```
product1.discountPercent = 40
```

Get an attribute

```
percent = product1.discountPercent     # percent = 40
```



How to call methods of an object

The syntax

```
objectName.methodName([parameters])
```

Call the getDiscountAmount() method

```
discount = product1.getDiscountAmount()
```

Call the getDiscountPrice() method

```
salePrice = product1.getDiscountPrice()
```

25. Describe the value of an ORM - Object Relational Mapper as found in Django.

ORM Speeds-up Development - eliminates the need for repetitive SQL code. Reduces Development Time. Reduces Development Costs. Overcomes vendor specific SQL differences - the ORM knows how to write vendor specific SQL so you don't have to worry about that.

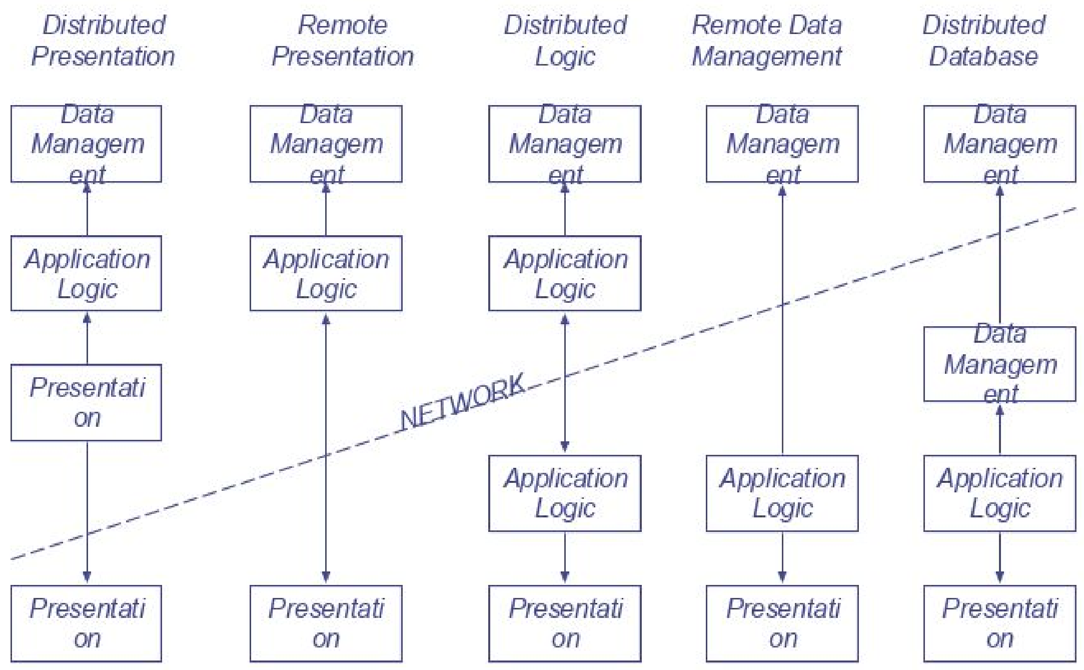
An **object-relational mapper (ORM)** is a code library that automates the transfer of data stored in relational databases tables into objects that are more commonly used in application code.

ORM provides a high-level abstraction upon a relational database that allows a developer to write Python code instead of SQL to create, read, update and delete data and schemas in their database.
It can speed up web application development, especially at the beginning of a project.
There are numerous downsides of ORMs, including

1. Impedance mismatch
2. Potential for reduced performance
3. Shifting complexity from the database into the application code
   Django's ORM works well for simple and medium-complexity database operations. However, there are often complaints that the ORM makes complex queries much more complicated than writing straight SQL or using SQLAlchemy.

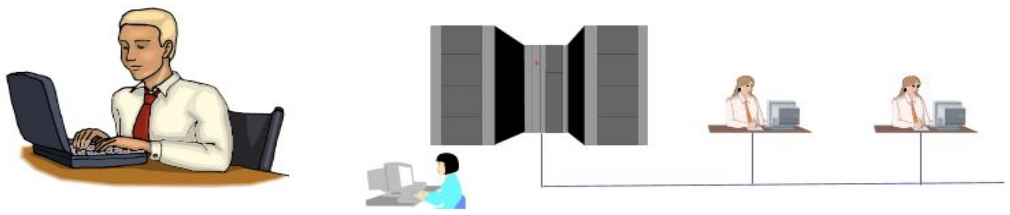
26. Be able to classify applications using the model of Computing: Distributed Presentation, Remote Presentation, Distributed Logic, Remote Data Management, Distributed Database. Also be able to determine the number of tiers in an application?

Classic Client Server Models – Gartner Group

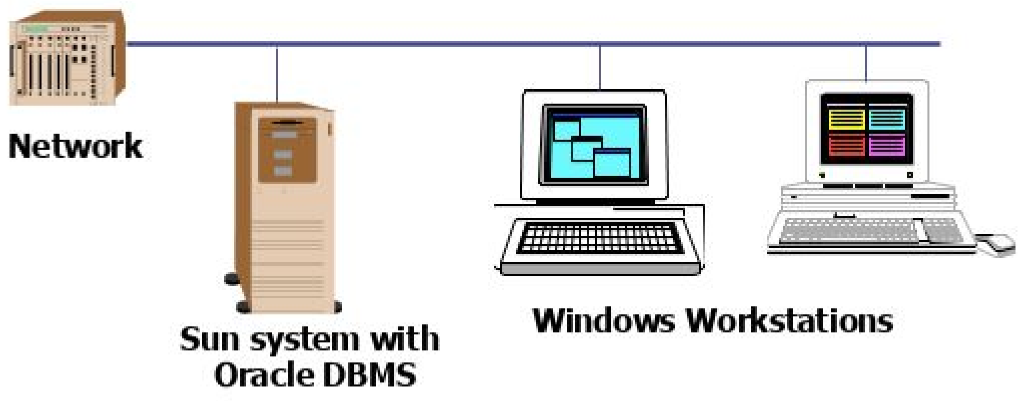


Distributed Presentation - Only presentation management function shared between client and server. Everything else remains on the server.
Remote Presentation - Presentation manager entirely on client. Data logic and data manager on server side.
Distributed Logic - Application is split into presentation logic and data logic components.
Remote Data Management - Database manager resides on server on server. presentation management and data logic reside on the client.
Distributed Database - Portion of the database resides on both client and server sides. DBMS manages communication involved.

1-tier : A user interacting with a personal computer. A user interacting with a terminal attached to a mainframe. It is the simplest one as it is equivalent to running the application on the personal computer. All of the required components for an application to run are on a single application or server. Presentation layer, Business logic layer, and data layer are all located on a single machine.

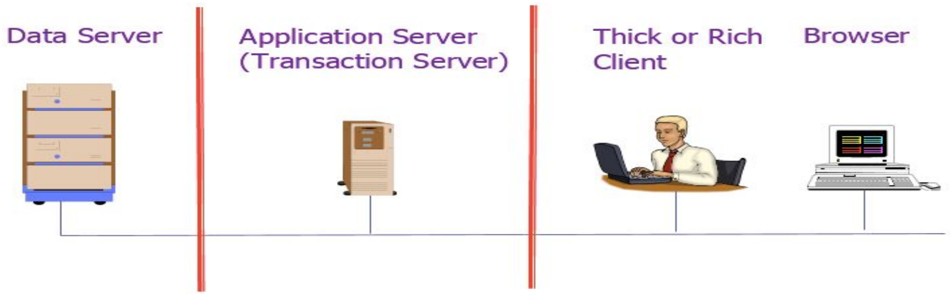


2-tier : It is like Client-Server architecture, where communication takes place between client and server. In this type of software architecture, the presentation layer or user interface layer runs on the client side while the dataset layer gets executed and stored on the server side. There is no Business logic layer or immediate layer in between client and server.



3-tier : **3-tier architecture** has three different layers. A third layer lies between the user and the data.

- Presentation layer
- Business Logic layer
- Database layer



27. Be able to classify middleware as asynchronous or synchronous and explain what this means.

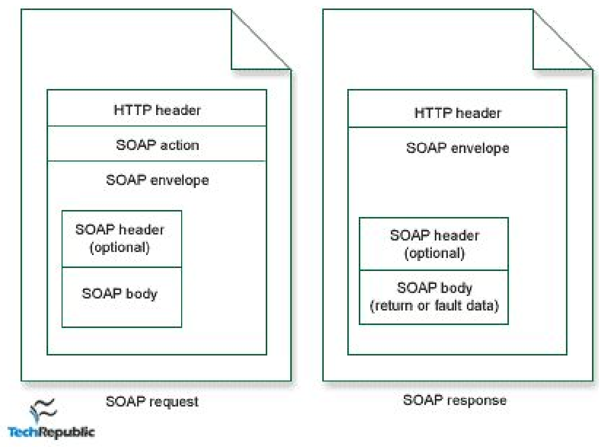
Asynchronous middleware - It allows two or more applications to send information between each other without being dependent on the other application for immediate processing.

Synchronous middleware - it is tightly coupled to applications. The applications are dependent on the middleware to process one or more functions at the remote application until processing can resume.

28. Explain how both SOAP and REST based web services work and differences between them. Contrast these web services with messaging queuing middleware like IBM MQ or Java Messaging Services

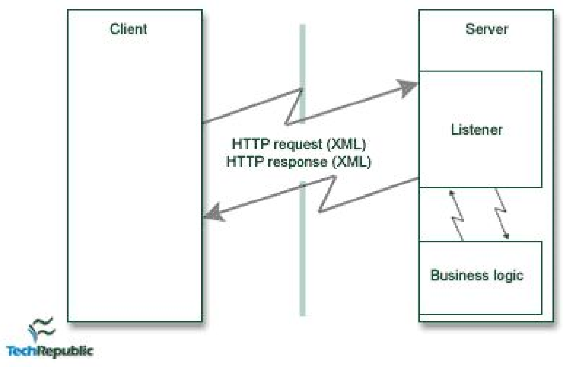
SOAP: Simple Object Access Protocol

SOAP primarily uses the standard HTTP request/response model.



The client wraps a method call in SOAP/XML, which is then posted over HTTP to the server. The XML request is parsed to read the method name and parameters passed and delegated for processing. The XML response is then sent back to the client, containing the return value—or fault data—of the method call. Finally, the client may parse the response XML to make use of the return value.

The server uses a "listener" to process SOAP requests. The listener is simply the server code at the specified URL for parsing the XML request, making the procedure call, and wrapping the result in XML to send as the response.



SOAP is used for calls across machines and networks.

REST: Representational State Transfer

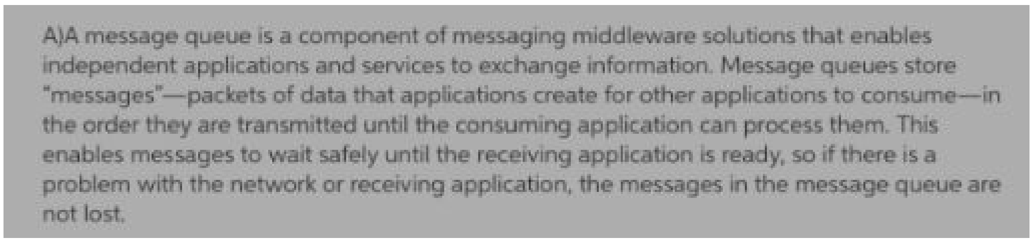
Use a URL for requests; response can be in one of several forms like CSV or JSON.

Soap Advantages

SOAP provides the following advantages when compared to REST:

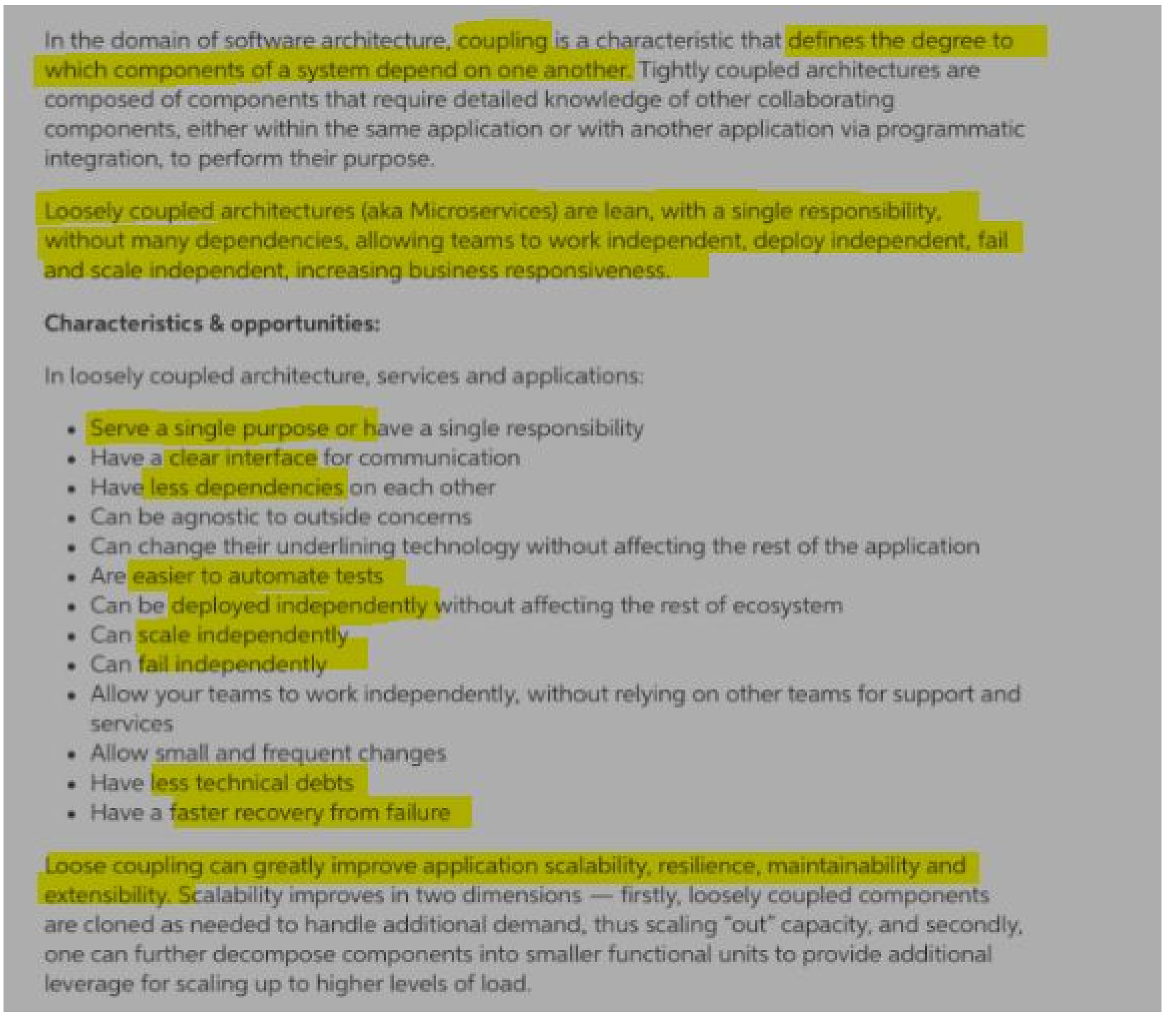
- Language, platform, and transport independent (REST requires use of HTTP)
- Works well in distributed enterprise environments (REST assumes direct point-to-point communication)
- Standardized
- Provides significant pre-build extensibility in the form of the WS* standards
- Built-in error handling
- Automation when used with certain language products

REST Advantages
REST is easier to use for the most part and is more flexible. It has the following advantages over SOAP:
- No expensive tools require to interact with the web service
- Smaller learning curve
- Efficient (SOAP uses XML for all messages, REST can use smaller message formats)
- Fast (no extensive processing required)
- Closer to other web technologies in design philosophy

Differences with message queuing middleware:
Definition of message queuing MW:

A)A message queue is a component of messaging middleware solutions that enables independent applications and services to exchange information. Message queues store "messages"—packets of data that applications create for other applications to consume—in the order they are transmitted until the consuming application can process them. This enables messages to wait safely until the receiving application is ready, so if there is a problem with the network or receiving application, the messages in the message queue are not lost.

Ideal for asynchronous transactions

29. Be able to explain the value of a loosely coupled software architecture


In the domain of software architecture, coupling is a characteristic that defines the degree to which components of a system depend on one another. Tightly coupled architectures are composed of components that require detailed knowledge of other collaborating components, either within the same application or with another application via programmatic integration, to perform their purpose.

Loosely coupled architectures (aka Microservices) are lean, with a single responsibility, without many dependencies, allowing teams to work independent, deploy independent, fail and scale independent, increasing business responsiveness.

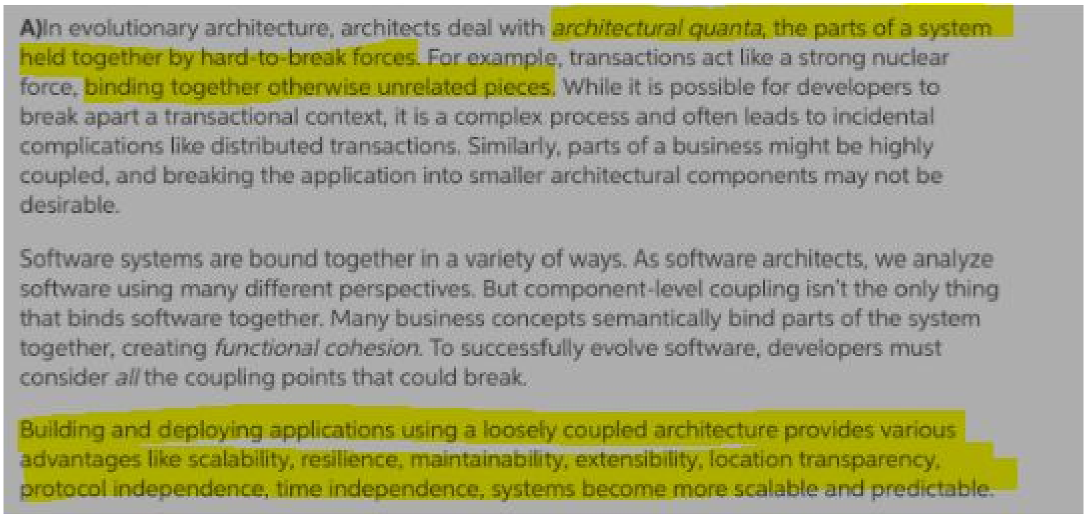
Characteristics & opportunities:

In loosely coupled architecture, services and applications:

- Serve a single purpose or have a single responsibility
- Have a clear interface for communication
- Have less dependencies on each other
- Can be agnostic to outside concerns
- Can change their underlining technology without affecting the rest of the application
- Are easier to automate tests
- Can be deployed independently without affecting the rest of ecosystem
- Can scale independently
- Can fail independently
- Allow your teams to work independently, without relying on other teams for support and services
- Allow small and frequent changes
- Have less technical debts
- Have a faster recovery from failure

Loose coupling can greatly improve application scalability, resilience, maintainability and extensibility. Scalability improves in two dimensions — firstly, loosely coupled components are cloned as needed to handle additional demand, thus scaling "out" capacity, and secondly, one can further decompose components into smaller functional units to provide additional leverage for scaling up to higher levels of load.

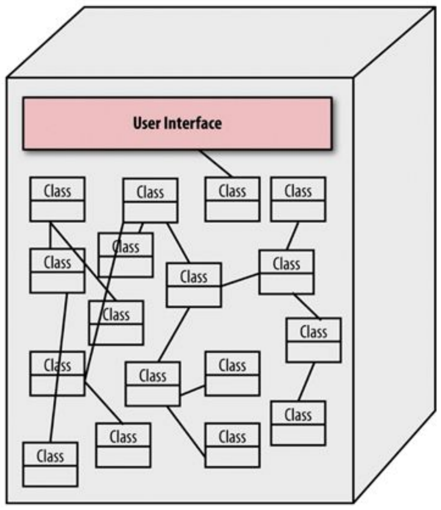
30. Explain what an architectural quantum is and how it applies to loose and tight coupling architectures


A)In evolutionary architecture, architects deal with *architectural quanta*, the parts of a system held together by hard-to-break forces. For example, transactions act like a strong nuclear force, binding together otherwise unrelated pieces. While it is possible for developers to break apart a transactional context, it is a complex process and often leads to incidental complications like distributed transactions. Similarly, parts of a business might be highly coupled, and breaking the application into smaller architectural components may not be desirable.

Software systems are bound together in a variety of ways. As software architects, we analyze software using many different perspectives. But component-level coupling isn't the only thing that binds software together. Many business concepts semantically bind parts of the system together, creating *functional cohesion*. To successfully evolve software, developers must consider *all* the coupling points that could break.

Building and deploying applications using a loosely coupled architecture provides various advantages like scalability, resilience, maintainability, extensibility, location transparency, protocol independence, time independence, systems become more scalable and predictable.

31. What is an unstructured monolith architecture and why is it difficult to change? (Week 1, Architecture Models) The unstructured monolith is tightly coupled, All classes, methods and components must be recompiled, tested and deployed together. This architecture is slow to change due to care needed to ensure code is not broken when new features are added. Also slow to retest.



32. Explain the value of a layered monolith or an unstructured monolith.(Week 1, Architecture Models)

The layered monolith has the addition of business rules and allows for changes in logic without recompiling and testing the entire system. Often business rules can be changed and tested and deployed to production in hours. (faster to test and deploy than the unstructrued monolith).

33. Explain how an ESB and services oriented architecture work and what value they bring to the services used in the application (week 2, Middleware, part II)

ESB = Enterprise Service Bus

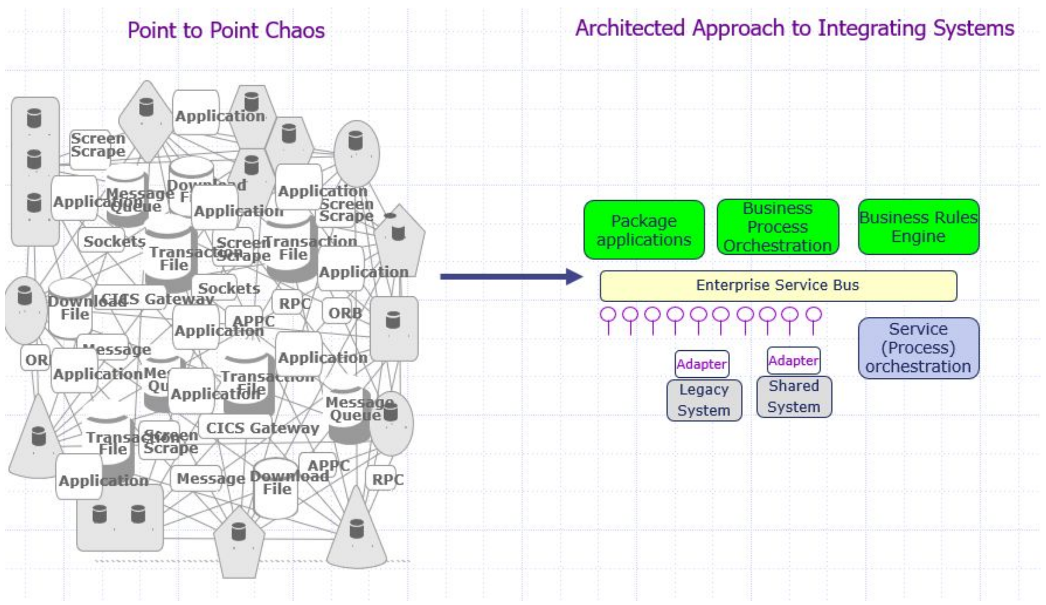
An ***enterprise service bus*** is a standard for integrating enterprise applications in an implementation-independent fashion, at a coarse-grained service level (leveraging the principles of service-oriented architecture) via XML-based messaging engine (the bus).

Ex: "Trading partners send key information between each other such as hospitals sending claim information to an insurance company or a supplier sending invoices to Walmart.

Use an ESB when need to integrate several different systems (e.g., connect one company's transactions to another company's transaction system). Ex: SAP has its own ESB

Pros of ESB and SOA: fast, easier to use (less customization), scalable

Cons: single point of failure, requires significant planning/skills



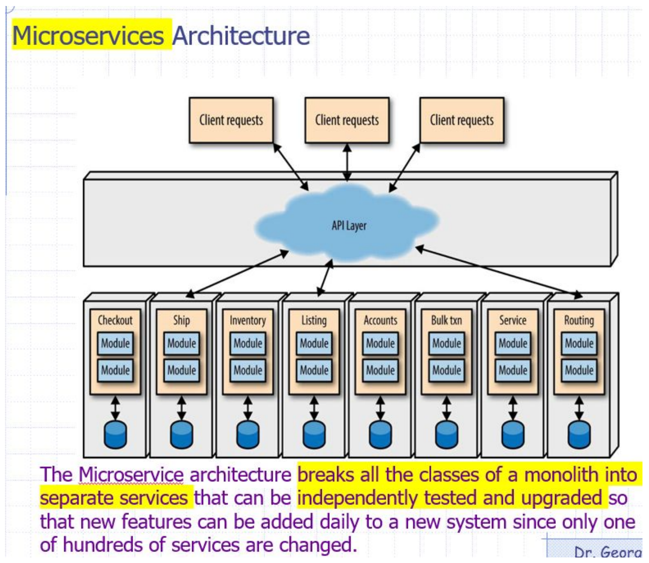
ESB allows the different loosely coupled clients to interact with each other and integrate with another component. ESB allows decoupling of clients with Service providers. Examples are: B2B products or services are examples of loosely coupled systems. Most legacy systems in the organizations are tightly coupled and monolithic.

Two business benefits are as follows:

1. Speed – With less dependencies the components can be built and tested faster.
2. Flexibility -Individual components/services can be customized as per the needs.

Pros of ESB and SOA: fast, easier to use (less customization), scalable
Cons: single point of failure, requires significant planning/skills

34. Explain what a microservice architecture is and how it enables loose coupling and the possibility of adding features on a daily basis to a major application. (Week 1, Architecture Models)


Microservices Architecture

The Microservice architecture breaks all the classes of a monolith into separate services that can be independently tested and upgraded so that new features can be added daily to a new system since only one of hundreds of services are changed.

35. Explain the concepts behind an Agile Architecture as defined by the Open Group

Agile architecture means how enterprise / system / software architects apply architectural practice in agile software development.

-Autonomous teams
-Customer experience is key
-Product Centricity
-?

36. Be able to describe what gRPC is and how it compares to standard REST based services. Why are companies using it instead of REST? (Week 2, Intro to Web APIs)

Definition: **gRPC** is an open source remote procedure call (RPC) system initially developed at Google. It uses HTTP/2 for transport, that use Protocol Buffers as the interface description language, and provides features such as authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts. It generates cross-platform client and server bindings for many languages. "Soap on steroids"

Different in how serialization is used. It is based on Protocol Buffers, an open source mechanism for serializing structured data, which is **language and platform neutral.**

Used by Facebook, GitHub, Paypal

Compares to REST:
-gRPC uses serial protocol buffers; REST uses JSON (text-based and larger)
-REST is better for browser support
--gRPC faster than REST

Why use? protocol buffers are faster, more efficient. Language & platform neutral. Best suited for a microservice architecture.

37. Be able to describe what GraphQL is and how it compares to standard REST based services. Why are companies using it instead of REST? (Week 2, Intro to web APIs)

GraphQL is a next-gen REST API.
Use because it gives more capabilities to client-side applications.Client decides what to fetch, not the server (data query language). Agnostic to underlying structure or programming languages.
Used by larger companies for massive performance (ex: Google)
