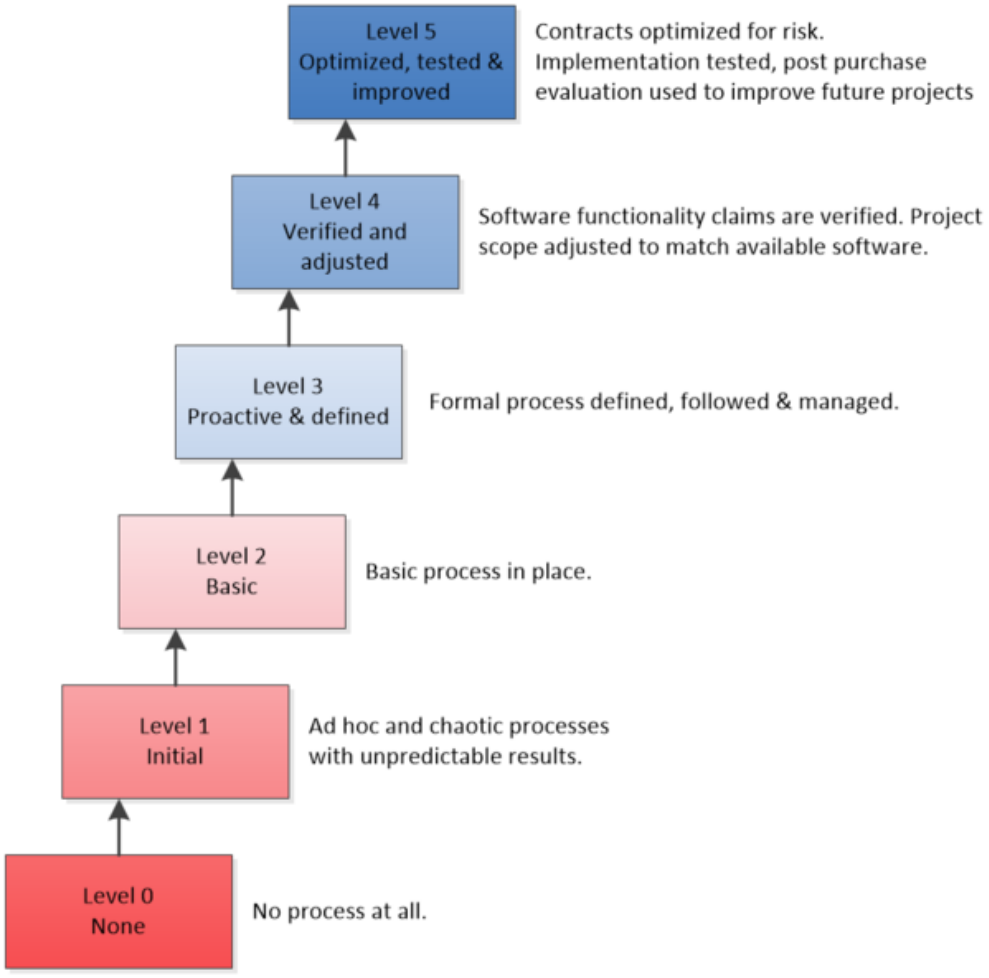
1	Describe the Software Selection Maturity Scale? What is its relationship to technology adoption in the Enterprise? Software Selection Maturity Scale (SSMS) is a 5-level scale which measures the maturity of a given enterprise for its technology evaluation and acquisition process. Many enterprises or their parts perform low down in the scale. The following shows the diagram of the above-mentioned scale  **Technology Adoption** Software technologies get adopted in a variety of into the enterprise but generally the enterprise is slow to adopt new technologies until the industry has been seen to adopt them - IT or vertical. Where solutions are insourced or outsourced, it is often the case that similar or competitor enterprises have already adopted the same solutions - introduces dependent technology needs. In house development tends to use solutions supported by large stable vendors and which have existed for years or even decades. Enterprises are often late majority or laggard adopters in the context of the information technology adoption lifecycle.	10 Marks

2 Compare and contrast the monolithic and SOA models of enterprise application software. Describe the benefits of a SOA composition approach to application construction. 10 Marks

Monolithic	SOA
Monolithic software is small number of large applications providing many diverse functions on a widely variety of data sources.	Service Oriented Architectures is a style of software design which constructs software solutions from technology independent components. These can be composed together over a network using some well-defined network protocol.
Problems arose such as the cost of maintaining large and complex systems. Lack of flexibility also existed in regards to providing solutions.	Contrasts to the monolithic in the sense that solutions to problems which the monolithic does not address can be created by the end user using the components.
Monolithic applications have been decomposed into smaller single purpose services.	Components that make up SOAs are known as services.
Difficult to test because a good test system in a large system is very hard and infeasible at times.	Easier to test in contrast to Monolithic model.

Benefits of SOA Composition

SOA is made up of services. Services have 4 basic properties:

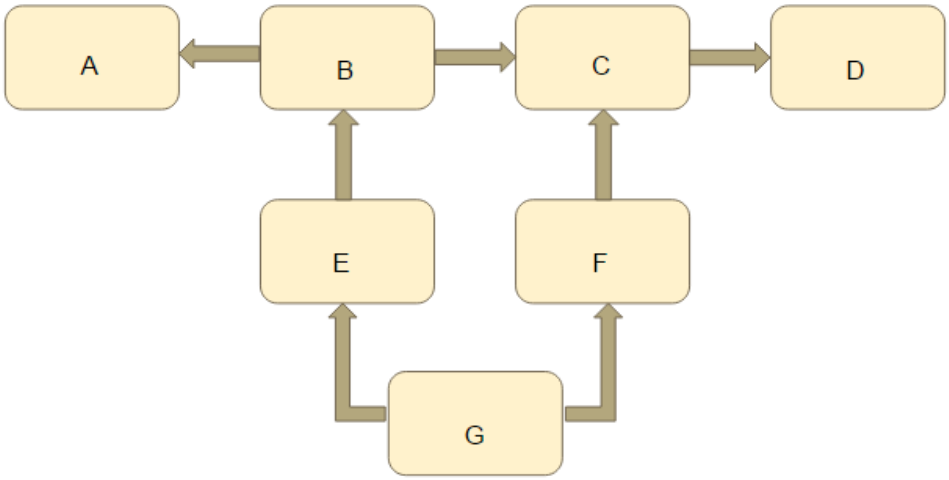
1. Logically represents business activity with a defined outcome
2. Self-contained
3. Black-box from a consumer's perspective
4. Composed from one or more other services

Hence services are easier to construct, maintain and test and offer considerable cost and time advantages.

Consumer expects the service to complete its task without the need for external dependencies. Hence service composition is a natural graph in nature with loose coupling.

SOA is independent of any implementation technology which leaves a real-world enterprise implementation potentially a wide variety of technology options to choose from.

The following diagram shows service/sao composition:


A
B
C
D
E
F
G

3	Why have enterprises moved towards web technologies for service software construction? What are the principal benefits? Web technology stack has become more popular for services implementation. HTTP is the favoured application-layer protocol and XML and JSON are the favoured serialisation formats. These have a wide community support and set of industry standards. Web stack is known to offer least costly solutions to common service construction concerns. Services which use web technologies don't necessarily need to be consumed on the web or by the web clients. The web is just a toolkit for the common building blocks. As businesses start to rely more on open standards and open source software components, enterprises are willing to contribute to the development and support of the above mentioned. Large technological companies are providing financial assistance and resources to the standard bodies and software projects. This will help to maintain their influence and their strategic technical direction.	10 Marks

4 What are the advantages and disadvantages of HTTP statelessness as the basis of a service application protocol tier? — 10 Marks

Advantages	Disadvantages
Requests are issued by the client and responded to by the server.	Clients and servers cannot make any guarantees about application level state at the HTTP level and hence making communication session less from this perspective.
Each request and response can use its own TCP/IP connection between the client and one or more servers.	Some sets of API calls are logically related and even transactionally grouped.
Connections between clients and servers can be proxied between one or more immediate nodes.	
Each API call to a service is logically independent from any other.	
Has a lot of standard support and strong community set of libraries.	
Great solution for both enterprise intranet and public intranet.	
De facto standard application layer protocol.	

5	Describe the architectural constraints of the REST architectural pattern **Client-Server:** Establishes the separation of concerns between the service provider and the service consumer. This is known to offer one or more capabilities and listens for requests for those capabilities. The consumer is responsible for presenting the responses to the user and taking any corrective actions on foot of errors. **Stateless:** States that no server-side state exists between any two REST requests from a consumer, i.e. requests are self-contained and standalone. **Cacheable:** Service responses can be explicitly labelled as cacheable or non-cacheable. This allows intermediating caching nodes to reuse previous responses to requests without having to go all the way back to the service. This is a key idea in making RESTful services scalable. **Layered:** An arbitrary number of nodes can be placed between the ultimate service and service consumer. Their existence must be fully transparent so that they can be added and removed at will. This allows for the distribution and scalability of RESTful service solutions in practice. **Uniform Contract:** This constraint states that all services and service consumers must share a single, overall technical interface. This is the primary constraint that distinguishes REST from other architectures. In REST, this is applied using the verbs (commands) and media types of HTTP. The contract is usually free from specific business logic context so that it can be consumed by as wide a variety of client types as possible.	10 Marks

6	Explain the relationship between resources, models and views? What is meant by view aggregation? **Resources** An independent representation of some service state or behaviour. In regards to API design, it is known to map resources onto server side models on a near one to one basis. **Model** A model is the name given to some abstraction of internally persistent service state i.e. a database table. The following is a good example of resources based on models. e.g. /customers/123 ➔ customer details for id 123 **Views** A model is the abstraction of the service state (e.g. a database table) but a view is the representation of the model. Views are representation of a model using some XML or JSON. When a resource is based on a model, the view also becomes the representation of the resource. Hence we come to the model view controller i.e. MVC. **View Aggregation** Consumers always want aggregated views of two or more related resources. There are two options to choose from: 1. Create a new resource on the server side which provides the aggregation required (e.g. some nested structure comprised of multiple resources or resource fragments) 2. Craft the aggregation on the client side by making whatever API calls are necessary to do so	10 Marks

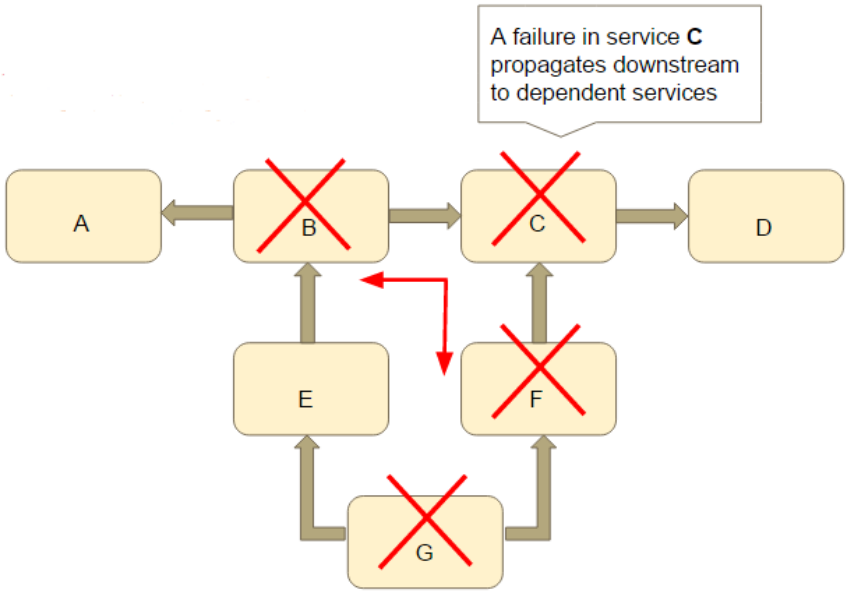
7	Describe the five RESTful operations, giving examples using HTTP. What is meant by idempotence? Mention which of the REST operations are idempotent and why.
	
	Idempotence
	This is the property of an operation which means an operation can be applied many times to a value without changing the outcome after its first application. In REST, an operation is idempotent if after the first operation, other applications of that operation don't alter the service state.
	
	POST ➔ used to create item resources having a message body containing attribute key/values to populate the newly created resource. POST is not idempotent as each successful call creates a new resource.
	
	e.g.
	curl -X POST https://api.example.com/customers/237324632/notices -d
	{
	"subject": "Account status",
	"body": "Dear john, please review your outstanding balance ...",
	"delivery": "urgent"
	}
	
	GET ➔ command is used to read resources and this is idempotent provided it is strictly implemented which respect to the resource state proper.
	
	e.g.
	curl -X GET https://api.example.com/customers/237324632/notices/213
	
	HTTP/1.1 200 OK
	{
	"subject": "Account status",
	"body": "Dear john, please review your outstanding balance ...",
	"delivery": "urgent"
	}
	
	PUT ➔ used to fully update a resource is an idempotent operation as multiple PUTs with the same values don't alter the state following the first one.
	
	e.g.
	curl -X PUT https://api.example.com/customers/237324632/notices/213 -d
	{
	"subject": "Account status",
	"body": "Dear john, please review your outstanding balance ...",
	"delivery": "normal"
	}
	
	PATCH ➔ used to partially update a resource but this is not considered to be automatically idempotent because of the way an API might do the updates.
		10 Marks

	e.g.
	curl -X PATCH https://api.example.com/customers/237324632/notices/213 -d
	{
	"delivery": "normal"
	}
	
	DELETE ➔ used to affect a resource removal and this is idempotent because once the item resource is removed it doesn't matter how many subsequent requests to remove it are made.
	
	e.g.
	curl -X DELETE https://api.example.com/customers/237324632/notices/213	

8	Explain the problem of failure propagation in SOA systems. What are the desirable characteristics of an API versioning system? What are the two kinds of API compatibility?	10 Marks

In an SOA environment, failures in upstream services may cause failures to propagate to downstream dependent services. One of the most common sources of errors occurs when changes to an API doesn't fully appreciate some of its behaviours being depended on by consumers, which includes bugs.

The following diagram shows failure propagation in SOA systems.



Versioning Systems

This is to provide a shorthand to API consumers regarding which features of the API are supported and how they work. APIs do change and that consumers need to be aware of those changes and can assess impact. Versioning schemes take many forms and which one is best depends on the circumstances and particulars of a service and its consumers.

A good scheme should convey the following information:

1. API stability - meaning how likely it is to change and be relied upon
2. Major changes - that new features have been added or existing ones changed
3. Minor changes - that existing features have been updated (e.g. bug fixes)
4. Build identifier - pinpoints the precise origins (contributing codebase) of the API version (often related to the underlying source code control system)

API Compatibility

Backward compatibility - changes to the API still allow legacy API consumers to transparently interoperate with the new version as if it was the old version (i.e. the client cannot distinguish between them). This type is usually critical.

	Forward compatibility - the API is designed in such a way that it will transparently interoperate with a future version of itself allowing clients using a new version of the API to work with legacy services at least to the extent of the functionality of the functionality offered by the legacy API.	

9	Describe the major elements of the logical data model. Describe how it abstracts the details of database access in the application tier. The Logical Data Model consists of the following 3 internal layers of abstraction: **Models** - Language-specific data structures which abstract the representation of the database entities (e.g. classes and objects) **Language bindings for SQL** - Generic API for interface from the middleware language to SQL services on the database (e.g. execution of queries) **Database vendor driver** - Implementation-specific logic for translating SQL bindings into the raw connection API (e.g. binary protocol for parameterised query data) **Abstraction** Models are the language-specific abstraction of the database entities. In RESTful API services, models provide the architectural bridging between database entities and the exported resource state and behaviour. In the simplest view, a model is just an internal memory representation of persistent data structures which can be validated, processed and represented independently of the database itself. In practice, models are implemented as classes and objects in OOP languages like Java.	10 Marks

10	Describe in detail the pathology of a SQL injection exploit. What should the application developer to avoid this kind avoid this kind of vulnerability. SQL injection attacks are a security exploit of vulnerable query generation practices in code. What the attacker wants to do is exploit a query formation vulnerability by repeatedly sending queries to the service with malformed input in the hope that he will find a flaw. Once a flaw is found, the attacker can potentially mount arbitrary attacks on the system to learn more about the schema, the data and extract or modify critical values. **Avoiding vulnerability** 1. Using pre-parser function which checks the validity of the query before it is executed (e.g. mysqli::escape_string from PHP) - not as secure in practice 2. Using prepared statement or parameterised queries 3. Using a stored procedure with typed arguments 4. Isolating the execution within a tight security sandbox using database privileges	10 Marks