|  |  |
| --- | --- |
| 1 | The Software Selection Maturity Scale is a five-level scale which measures the maturity of a given enterprise for its technology evaluation and acquisition process.  Many enterprises or parts thereof perform low down in the scale, these include…  1. Initial -> 2. Basic -> 3. Proactive & Defined -> 4. Verified & Adjusted -> 5. Optimised Tested & Approved  The SSMS describes the adoption of new technologies as it goes through the various phases of what new software adoption would entail. These include the initial being a chaotic process to the basic process being in place then a formal process defined followed and managed to functionality claims are verified and scope adjusted to match available software and finally contracts are opitimised for risk, implementation, test and post evaluation. |
| 2 | Monolithic software tends to be a small number of a large application providing many diverse functions on a wide variety of a data sources. A problem with monolithic software for the business is that the cost of maintaining such a large and complex systems and lack of flexibility to be able to provide new or bespoke solutions in timely manner.  On the other hand Service-Oriented Architectures is designed to see the construction of software solutions from a set of technology-independent components which can be composed together over a network using some well-defined network protocol. This contrasts with monolithic application in the sense that the solutions to the problems not originally addressed by the monolith can be created by end-users more flexibly from the components.  Using SOA the application is split into a number of technology-independent components or services, which allows for high coupling and cohesion throughout the application. This makes it ideal for application construction. |
| 3 | Enterprises have moved towards using web technologies for service software construction because they are compatible with all programming languages, the use of cloud computing removes the need for any In house tech, and it is highly standardised making API’s easily integrated for constructed for 3rd party use.  Another reason why enterprises have moved to using web technologies is the shift to client-side programming for designing web applications as it allows for other applications to share their connection to the services. |
| 4 | Advantages:   * RESTful web services work seamlessly with HTTP * Easy to identify each request and each one is independent * Simplifies the server design because there is no dynamically allocated storage * If a client dies in mid-transaction, no part of the system needs to be responsible for cleaning up the present state of the server   Disadvantages:   * Data or information sent via HTTP is not encrypted * Web services need to get extra information in each request * The way it handles sessions with HTTP is adding more information every time to the request like a cookie |

|  |  |
| --- | --- |
| 5 | The Achitectural constraints of REST can be divided into its Client-Server, Stateless, Cacheable, Layered and Uniform Contract architectural constraints.  Further explained:  Client-Sever:  Establishes the separation of concerns between the service provider and the service consumer. The provider offers 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:  The principle that no server-side state exists between any two REST requests from a consumer, i.e. requests are self-contained and standalone. Contrast this with a transactional style, for example  Cacheable:  Service responses can be explicitly labeled 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 |
| 6 | Effectively when a resource is based on a model, the view also becomes the representation of the resource.  Model-view-controller (MVC) API builder frameworks take advantage of the model-and-view-as-resource convention to allow the automated generation of server-side API boilerplate code  View aggregation is two or more related sources being compressed into a single view. Aggregation on the client has the advantage that exactly what is needed can be crafted for the consumer that needs. However a downside is that usually there is a performance penalty to having making multiple API calls to fetch the necessary resources and adds more complexity to the API implementation.  View aggregation can be implemented in two ways:   1. Creation of a new resource on the server side which provides the aggregation required 2. Crafting the aggregation on the client side by making whatever API calls are necessary to do so |
| 7 | Five Restful Operations:   * POST * Create a new resource * GET * Read resources * PUT * Fully update resource (Works somewhat as a replace as data can be deleted). * PATCH * Partially update resource (data cannot be deleted unlike PUT). * DELETE * Remove a resource   Idempotence is the property of an operation such that the operation can be applied multiple times to some value without changing the outcome beyond its first application.  Idempotence operations include GET, PUT and DELETE. |
| 8 | In an SOA environment, failures in upstream services may cause failures to propagate to downstream dependent services.  Desirable characteristics of an API versioning system include:   1. API stability  * How likely is it to change and be relied upon  1. Major changes  * That new features have been added or existing ones changed  1. Minor changes  * That existing features have been updated  1. Build identifier  * Pinpoints the precise origins of the API version   API compatibility:   1. 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.  1. 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 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 | The middleware tier is responsible for implementing the logical data model which is an abstraction of the database entities, relationships and data representation.  The LDM itself comprises of a model layer, SQL binding layer and a database driver.  The RDBMS LDM comprises of typically three internal layers of abstraction:   * Models * Language-specific data structures which abstract the representation of the database entities * Language binding for SQL * Generic API for interface from the middleware language to SQL services on the databases * Database vendor driver * Implementation-specific logic for translating SQL bindings into the raw connection API |
| 10 | Pathology of an SQL injection exploit is when an attacker wants to exploit query formation vulnerability by repeatedly sending queries to the service with malformed input in the hope that they will find a flaw.  After a flaw is found the attack can potentially mount arbitrary attacks on the system to learn more about the schema, the data and exact or modify critical values.  These vulnerabilities can be eliminated through using:   * Pre-parsed function which checks the validity of the query before it’s executed * Using prepared statement or parameterised queries * Using a stored procedure with typed arguments * Isolating the execution within a tight security sandbox using database privileges |