**Name: Ronan Connolly**

**Student Number: C15737505**

**Q1. Describe the Software Selection Maturity Scale? What is its relationship to technology adoption in the Enterprise?**

The software selection maturity scale is a measurement consisting of five levels. The scale measures the maturity of an enterprise’s technology and evaluation and acquisition process. The five levels are:

1. Initial
2. Basic
3. Proactive and Defined
4. Verified and Adjusted
5. Optimised, Test and Approved

Enterprise Systems are an essential and critical part of modern business and any change could potential entail high risk to the smooth operation of the business. Enterprises are generally slow to adopt to new technology until they have been thoroughly tested in the industry and other companies have adopted them. Solutions for enterprise software often consist of using third party vendors who provide a stable and long running software solutions.

Business’s looking to adopt a new technology or enterprise system should carry out a detailed acquisition process consisting of the following steps:

* Return on investment estimate
* Requirements and risk analysis
* Research available options and vendors
* Request Proposals from vendors
* Evaluation of the vendors proposals
* Verification of the vendors proposal and claims
* Project Management
* Develop legal contracts, licensing and support agreements

**Q2. Compare and contrast the monolithic and SOA models of enterprise application**

**software. Describe are the benefits of a SOA composition approach to**

**application construction.**

Monolithic models consist of a single system which contains and executes the entire enterprise logic. It contains a single code repository that is a managed by various teams. This model is restricted to how much it can scale. The entire application must be duplicated in order scale. This can lead to wasted resources as not every part of the application may be in demand, but these less popular parts of the must also be scaled even if they are not being utilized as much. The cost of maintaining a monolithic system is quite large and there is a lack of flexibility available while using such a system.

Service orientated architecture in contrast takes the monolithic application model and decomposes it into smaller single purpose functionality. Each service performs one task and communicates via message passing with other services. Services can be combined to create more complex solutions. Communication between services can be achieved using the HTTP protocol and JSON as a data serialization format. Using a standard serialization format allows each service to be language and technology independent.

Services should:

1. Represent a single business activity
2. Be completely self-contained
3. From the user/consumers perspective be viewed as a black box
4. Be itself composed of multiple other services if need be

**Q3. Why have enterprises moved towards web technologies for service software**

**construction? What are the principal benefits?**

Enterprises have moved towards using web technologies for services as there exists a set of industry standards regarding HTTP, XML and JSON, which have become the favoured protocol and serialization formats. There exists a large community to support these tools also.

The emergence of cloud computing has also prompted enterprises to move towards web technologies as it provides a means for the enterprise to no longer need to manage their own servers in house which can be very costly. Using cloud providers to manage the infrastructure of an enterprise application can reduce cost regarding lower payroll costs for maintenance and the “pay for what you use” model. However, a potential downside is that the enterprise may lose organisational knowledge of critical systems.

**Q4. What are the advantages and disadvantages of HTTP statelessness as the**

**basis of a service application protocol tier?**

HTTP is a stateless protocol where each request and response are an independent connection over TCP/IP. A stateless protocol provides high scalability, a request to be sent to any endpoint and endpoints can be added or removed when needed. Availability is high with a stateless protocol as if a request fails there is no state lost, simply resending the request is acceptable. Response can also be cached for higher speeds as there is no state.

However, calls to an API can be related in a transactional based system. For example, a shopping cart service requires storing what users have added to their cart. Each client and cart must be tracked. In these situations, state must be maintained. These can be achieved by using tokens and sessions.

**Q5. Describe the architectural constraints of the REST architectural pattern**

Representational State Transfer is an architectural pattern which abstracts service state and behaviour. Service state and behaviour is modelled as resources which are exposed via an API which consumers interact with. Message passing is used to interact via URI schemes which are unique, over the HTTP protocol. There is no representation of state in the REST architecture pattern and the actual internal state is independent of the resource representation exposed to the consumers.

REST uses the client-server model. The server is responsible for accepting and processing requests while the client is responsible for the presentation of the responses to the end users.

REST is stateless, no server state should exist between each request. Is request is required to be standalone and self-contained. This contrasts with the more traditional transactional style of operation.

REST should be cacheable. Response’s from the server can be marked as cacheable or non-cacheable. This can increase performance and reduce the need to make multiple requests. This allows the REST architecture to scale well.

REST should provide a way to be layered. N number of nodes should be able to be placed between the service and the consumer. These nodes must be transparent and be able to be added or removed when needed.

Finally, all services and service consumers must adhere to a Uniform Contract, which is the single, overall technical interface.

**Q6. Explain the relationship between resources, models and views? What is meant**

**by view aggregation?**

A resource is an independent representation of a service state or behaviour. A model is a mapping of a resource to a server-side model. This is a one to one mapping. A model is an abstraction over some persistent state such as a database table.

An example of a resource based on a model:

*/customers/123*

This would provide the details for the customer with the id 123.

A view provides the representation of the model. Data serialization formats such as JSON or XML are used to create the view for the end user to interact with. Views can be aggregated by creating a new resource on the server to provide the aggregation or construct the aggregation on the client side by making multiple calls to the API to retrieve the data required for the aggregated view.

Aggregated views on the server can provide the aggregation to all consumers which can be considered a benefit. However, this adds more complexity in the API design, coding and maintenance. Aggregating on the client side requires making multiple calls to retrieve the data needed, this can be an advantage of the client getting exactly what it needs to create the view. However, the downside is a performance hit as multiple calls need to be made.

**Q7. Describe the five RESTful operations, giving examples using HTTP. What is**

**meant by idempotence? Mention which of the REST operations are idempotent**

**and why.**

The POST operation is used to create a new resource. POST operations are not idempotent as each request will create a new resource.

*curl -X POST* [*https://someurl.com/customers/11111111*](https://someurl.com/customers/11111111) *-d*

*{*

*“subject”:”new customer”,*

*“body”: “………”*

*}*

The GET request is used to read a resource using their unique identifier. Reading a resource should not update the resource and by this account are idempotent.

*curl -X GET* [*https://someurl.com/transactions/121*](https://someurl.com/transactions/121)

*HTTP/1.1 200 OK*

*{*

*“subject”:”new customer”,*

*“body”: “………”*

*}*

The PUT operation is used to update an already existing resource. The resource can be updated either fully or partially. PUT is used to fully update a resource while PATCH is used to partially update a resource. An update that replaces an attribute in place would be considered idempotent. However, if the update involved adding an item to list it would not be considered idempotent.

*curl -X PUT* [*https://someurl.com/transactions/121*](https://someurl.com/transactions/121) *-d*

*{*

*“subject”:”new customer”,*

*“body”: “………”,*

*“deliver”: “normal”*

*}*

*curl -X PATCH* [*https://someurl.com/transactions/121*](https://someurl.com/transactions/121) *-d*

*{*

*“deliver”: “normal”*

*}*

The DELETE operation is used to remove an already existing resource. The resource URI is used to identify the resource to be removed. The DELETE operation is idempotent as once the resource is deleted, no other subsequent calls to remove it will work.

*curl -X DELETE* [*https://someurl.com/transactions/121*](https://someurl.com/transactions/121)

Idempotence is an operation that can be applied multiple times to some value without changing the outcome beyond the first operation.

**Q8. 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?**

Failure in upstream services cause failures in downstream services which dependent on the upstream services. This is usually caused by changes to the API don’t consider the services which depend on certain behaviours or operations in the service providing the API. Services that are built to consume other services can often be built based off the current behaviour of the service provider. However, the service provider may have bugs and in the process of fixing them lead to the dependent service breaking as the dependent service was built to interact with the service provided, bugs and all.

A failure in one service can propagate down through various services that depend directly or indirectly on it.

A versioning system can be used to show available API features that are supported to consumers. Versioning facilitates changes in services, allowing consumers time to migrate to new versions if needs be. Desirable characteristics of a versioning system include conveying the following to the consumer:

* API stability: How likely is the service likely to change
* Major Changes: New features have been added or existing ones changed
* Minor Changes: Existing features have been updated.
* Build Identifier: The API version relating to the source code control system.

There are two kinds of API compatibility:

Backwards Compatibility, where changes to the API still facilitate older and legacy consumers to work with the new version transparently. The client views this as the older version and cannot distinguish between new and old.

Forward Compatibility, where the API design allows for transparent operation with future versions of itself, allowing clients to use a new version of the API to work with legacy services.

**Q9. 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 abstracts the logic related to the database and its entities to language-specific models. This model should be independent of the database implementation. How the database handles queries should be hidden from the developer. This can be achieved using an Object Relational Mapper with maps databases elements to OOP constructs. For example, a database table can be mapped to a class, a row to an instance of a class, an attribute to an variable of a class and SQL queries to class or instance methods.

The Logical Data Model consists of Models, Language Bindings for SQL and Database Vendor Driver.

Models are language specific abstraction of the database entities. Models are implemented as classes in object orientated languages but can be viewed as a n internal memory representation of persistent data structures which are validated, processed and independent of the database.

SQL Language Bindings provide generic facilities for languages to interface with SQL on a target database system. Queries, parameters and results are represented in a standard way.

The Vendor Driver is a layer of vendor-specific details that abstract details of the database interface. These include the communications protocol and connection management, user access and credentials, connection pool management, thread safety, result representation and query string and parameter representation. These drivers can generally be swapped to another database driver without affecting the logical data model.

**Q10. 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 statements dynamically generated before reaching the database. This provides a potential security hole where an SQL injection attack can occur. These attacks can cause huge amounts of damage and impact on a business. Information could be deleted or changed by the attacker.

The attackers can exploit the security vulnerability by repeatedly sending queries with malformed input to the service. The attacker is looking for a flaw in the system that can be used to learn more information about the database such as schema, data, users and possibly assume the role of a user with privileges to delete or update data.

The developer should work toward eliminating the vulnerabilities. To prevent attackers from injecting bad code before the query planner parsers the statement the developer should:

* Use pre-parser function to check for validity of the query before its executed.
* Use a prepared statement or parameterised queries.
* Use a stored procedure with typed arguments.
* Use a sandbox to isolate the execution.