**Service Oriented Architecture**

Joshua Harrison

838252

Febuary 2018

**Discussion Points**

**Access control**

When a logged in user requests a resource, should a database call be made to validate the user has access to that resource (e.g. they’ve enrolled?)

OR on login should a

**Introduction**

For this project a MooC service in which universities can provide course material as both standalone units and as part of a wider qualification has been designed. A service orientated architecture(SOA) has been used to design the *‘MightyMooC’* platform. Erl(2010) defined eight core Service Oriented Design Principles as:

* ***Standard Service Contract****: Services must adhere to a standardized shared contract in order to promote standardization of design and ease of adoption.*
* ***Service Loose Coupling –*** *The edges between services should be minimally weighted, with no one service too heavily reliant or any other in order to perform its function.*
* ***Service Abstraction –*** *Service should be coarse grained with details of the internal logic of the abstracted from external consumers.*
* ***Service Reusability –*** *Services should be significantly agnostic so that they are candidates from multiple disparate business processes.*
* ***Service Autonomy –*** *Services should have significant control over their runtime environments and resources required for perform their function. This is in line with containerization and if taken to its logic limit, a microservices architecture.*
* ***Service Statelessness –*** *State information should not be handled by the services runtime in order to enhance resource scalability.*
* ***Service Discoverability –*** *Services should be discoverable through defined discovery mechanisms in order to maximize service reuse and thus avoid rewriting existing service logic*
* ***Service Composability –*** *Individual services can be grouped into ‘compositions’ with other services in a service inventory to solve high level business problems.*

Erl(2010) reasons that interoperability is the fundamental concept of Service Orientated Architecture as achieving interoperability leads to demonstrable return on investment (RoI) through the continual application and reuse of services in multiple business processes, both internal and external to the service owners. The eight principles highlighted above serve to drive the overall architecture towards interoperability. The MightMooC project has been design with these principles in mind, and their application shall be discussed in greater detail in the follow sections.

**Section A – Internal architecture**

This section shall focus on the design and implementation of the internal architecture and supporting business logic defined for MightMooC.

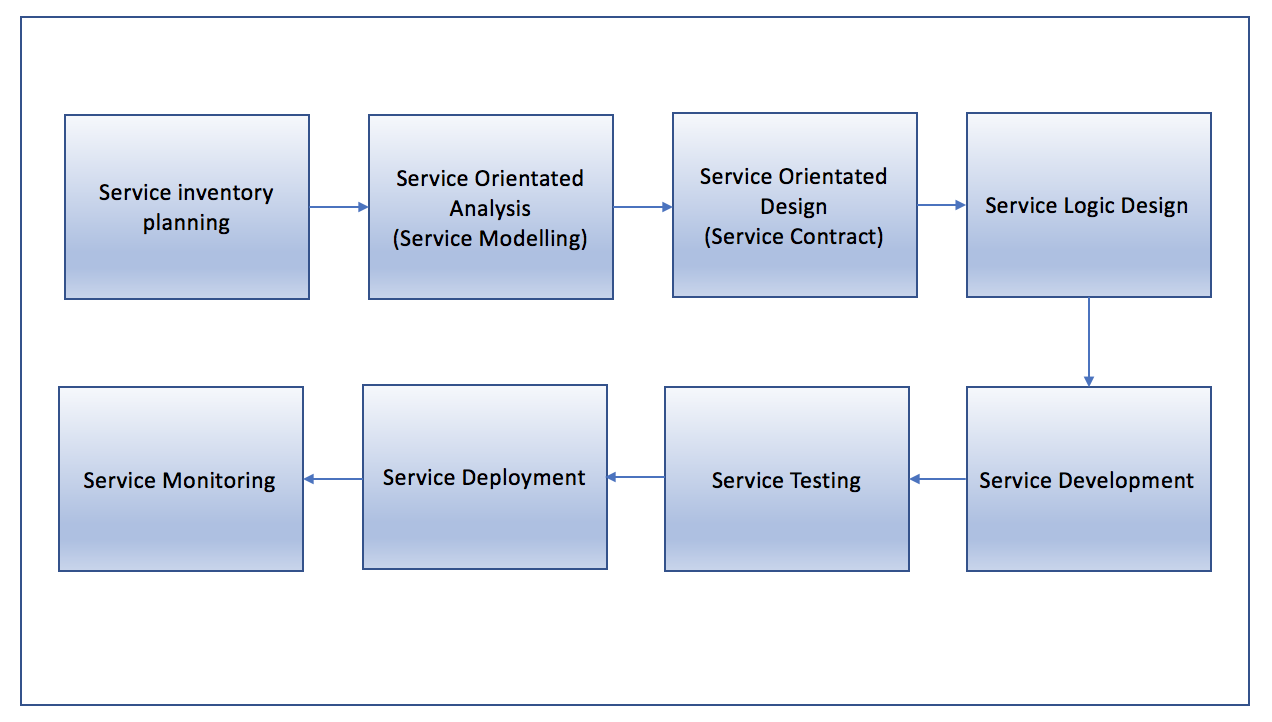
**A.1. Design Approach**

The project has been designed using a *'top down’* modelling implementation, meaning that before prior to code creation and schema definition, a prerequisite step is to analyse and deconstruct the central business requirements to the problem. This is approach is considered a best practice for SOA projects as it promotes enables a global definition of services and their capabilities and logical boundaries to be defined, leading to increased ubiquity, reduced code overlap between service, and crucially increased interoperability and composability.

There are caveats that can limit the success of implementing a top down design that should be considered. Namely, as a project progresses it naturally evolves, unexpected challenges and edge cases can arise, and new functionality and consumers can be added. The planning and documentation generated from the top down design need to be sufficiently flexible to facilitate the organic evolution of a project. A tightly scoped top down project will only succeed if its implementation is unobstructed by inflexibility and rigour.

A secondary concern is speed of development iterations, which can suffer as a result of top down design. This is a logical assertion as with a top down design process comes the overhead of increased pre-build requirements. A top down can be characterised as front loading the required man hours due to the greater emphasis placed on problem decomposition, analysing required service models and defining service contracts (Erl, 2010, pg. 94). However, the alternative allowing developers to build first and define contracts and documentation after or during the build process (bottom up design) for the gain of quick iteration and agile development comes with a high probability that the code produced will be less interoperable to both internal and external service consumers.

Figure 1 depicts the top down design approach that that is utilised by this project. This design has been adapted from Erl’s original specification ‘SOA project and lifecycle phases’ (2010, pgs 92-103), and as such phases related to the wider business context of SOA, such as organizational buy-in/adoption, service discoverability and versioning have been omitted.



***Figure 1: Project design phases***