Lesson 1: SWA overview

* What is software architecture

The fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution

* Why is software architecture important

Without a proper architecture the system might be

* + Difficult to build
  + Difficult to test
  + Difficult to maintain
  + Difficult to change
  + Difficult to understand
  + Difficult to reuse
  + Difficult to integrate with other systems
* Difference software architecture and software design
  + Software architecture oversees the whole while software design mostly focusses on one design problem
  + Software architecture needs to make sure that all requirements are met
  + Software architecture is often more at a higher level than software design
* What makes software architecture so difficult?
  + Many possibilities
  + Many frameworks, libraries, styles, patterns, databases, etc
  + Every possibility has advantages and disadvantages
  + Everything is a tradeoff
  + Everything changes all the time
  + Many decisions to take
* Difference architecture in a waterfall and agile project
* Characteristics of software qualities
* Architecture principles
  + Most important architecture principles
  + Keep it simple
  + Keep it flexible
  + Loose coupling
  + High cohesion, low coupling
  + Separation of concern
  + Information hiding
  + Principle of modularity
  + Open-closed principle

Lesson 2: Layering & Spring boot

* Different layers and their corresponding classes
* Service class
* Client centric vs server centric web frameworks
* Data access: stored procedures, JDBC, ORM
* Integration possibilities and their characteristics
* Spring boot
* Dependency injection @Autowired
* @RestController
* @Service
* @Component
* @Repository
* REST: GET, PUT, POST, DELETE : idempotent
* RestTemplate
* JMS : jmsTemplate and @JmsListener  
  You need to be able to write a simple Spring Boot application including the necessary classes and  
  their annotations and the necessary dependency injection.

Lesson 3: Domain Driven Design

* Principles of DDD (refer to slide)
  + Use one common language to describe the concepts of a domain
    - Common language means different people from the business use different names for the same thing
    - Ubiquitous language used by the team to capture the concepts and terms of a specific core business domain.
      * Used by the people
      * Used in the code
      * Used everywhere
  + Create a domain model that shows the important concepts of the domain
    - Rich domain model
    - Model: More complexity -> More modeling
      * Higher level of abstraction
      * Allows for visualization
      * Vehicle of communication
    - Domain model
      * Simplification of reality
      * Area of interest
  + Let the software be a reflection of the real world domain
* Anemic and rich domain model
* Orchestration and choreography
* DDD patterns: entity, value object (and its characteristics), domain service and domain event  
  You need to be able to design a rich domain model in UML with StarUML where you specify for each  
  domain class what DDD type it is

Lesson 4: Databases

* Scaling databases  
  o Horizontal, vertical  
  o Scaling load (\*)  
  o Scaling data (\*)  
  o Sharding  
  o Replication  
  o Brewers CAP theorem (P23)
* Relational databases  
  o Characteristics, Advantages and disadvantages  
  o Problems of relational databases
* Key-value store(Redis)  
  o Characteristics, Advantages and disadvantages
* Document store (mongodb)  
  o Characteristics, Advantages and disadvantages
* Column family store (cassandra)  
  o Characteristics, Advantages and disadvantages
* Graph database (neo4j)  
  o Characteristics, Advantages and disadvantages
* When do you choose which database
* How is the data structured in each of these databases

Lesson 5: Component based design

* What are components
* Why do we need components
  + High cohesion, low coupling
  + Flexibility
  + Reuse?
  + Decompose the domain in functional components
* API design best practices
* Component design
* DTO’s  
  You need to be able to design one or more components in UML with StarUML

Lesson 6: SOA and spring integration

* Hub and spoke  
  o Characteristics, advantages, disadvantages
* SOA  
  o Characteristics, advantages, disadvantages
* Integration patterns  
  You need to understand all given patterns. You do not need to draw the given pictures, but you need  
  to understand them.
* Spring integration  
  You need to understand the concepts of how spring integration works.  
  You need to be able to understand the XML configuration for the following patterns  
  • Service activator  
  • Gateway  
  • Channels  
  • Point-to-point vs. Publish-subscribe  
  • Synchronous vs asynchronous  
  • Custom router  
  • Filter  
  You do not need to write XML configuration (but you need to understand a given XML file).  
  You do need to know how to write Java implementation of the following patterns:  
  • Custom router  
  • Filter

Lesson 7

Monolith: advantages and disadvantages  
Characteristics of a microservice  
Advantages and disadvantages of a microservice  
Microservice and database  
Microservice and UI, micro-front-ends  
Microservice boundaries  
Domains: core, supporting and generic  
Microservice in the organization:  
devops team  
Conways law  
Feign  
Registry  
You do not need to write feign or registry code or configuration. You do need to understand the core  
principles of these techniques.

SCI

|  |  |
| --- | --- |
| **Lesson 1: SWA overview** | |
| Software architecture is defining all the important stuff in a software development project | The human physiology has the same structure as the structure of the Veda and Vedic literature  who are expressions of the structure of pure consciousness. |
| Software architecture is never ideal. We  have to find the right balance between the  different software qualities and architecture  principles | Nature always takes the path of least resistance so that the perfection of the unified field can  express itself in the relative creation |
| **Lesson 2: Layering & Spring Boot** | |
| An enterprise back-end system is typically divided in different layers. | Life is found in layers. |
| Dependency injection is a flexible technique to connect objects together by configuration. | Everything in creation is connected with everything else in its source, the Unified Field, the home of all the laws of nature. |
| **Lesson 3: Domain Driven Design** | |
| A rich domain model helps in communicating complex domain knowledge. | The daily experience of pure consciousness helps  in a more happy and successful life. |
| Instead of a large entity class, we strive for a small and simple entity class with many value classes | The Unified Field contains all knowledge in its simplest and most abstract form. |
| **Lesson 4: Database** |  |
| When your data does not fit on one node you automatically will get Data duplication. | Data duplication over multiple nodes means eventual consistency. |
| **Lesson 5: Component Design** |  |
| Components are encapsulated and completely autonomous plug-and-play elements. | The human nervous system is capable to transcend to that abstract field of pure consciousness which lies at the basis of the whole creation. |
|  |  |

**Why is software architecture important**

Without a proper architecture the system might be

* Difficult to build
* Difficult to test
* Difficult to maintain
* Difficult to change
* Difficult to understand
* Difficult to reuse
* Difficult to integrate with other systems

**Difference software architecture and software design**

* Software architecture oversees the whole while software design mostly focusses on one

design problem

* Software architecture needs to make sure that all requirements are met
* Software architecture is often more at a higher level than software design

**What makes software architecture so difficult?**

* Many possibilities
* Many frameworks, libraries, styles, patterns, databases, etc
* Every possibility has advantages and disadvantages
* Everything is a tradeoff
* Everything changes all the time
* Many decisions to take









