OLA 2 Systems Integration

Distributed Architecture – Modern Enterprise Architecture

Q1. Enterprise Architecture is used when dealing with large complex systems. There are four different domains: Business architecture, data architecture, application architecture and technology architecture. Different people specialize in the four different areas. A business architect specializes in understanding the people and the structure of the organization. Application architects specialize in applications that support and help manage the organization. Data architects focus on understanding the data how its acquired and which parts of the organization it supports. A technology architect understands the infrastructure where applications are hosted what data is on the cloud and what data is at the data centers.

These jobs all require different skills, and you can't expect one person to know all of them. Therefore, there are four levels of proficiency. Data architect would need level 4 proficiency in data but not need level 4 proficiency in something like leadership. Different roles for different people with different strengths. Within Enterprise Architecture there are different levels: strategic architecture, segment architecture and capability architecture. Solution architecture is the lowest in the levels but is not a part of Enterprise Architecture. Solution architecture is about finding specific products and solutions that you need for your organization. Enterprise architecture might suggest solutions, but solution architecture picks specific solutions. Enterprise architecture focuses on the big picture of the organization. A solution architect might design diagram and instructions as well as work closely with technical architects to ensure that the project is successful.

Q2. Team Topologies is about how teams collaborate and focusing on the team first when it comes to developing products. The book focuses on large organizations and systems. They define four types of team stream-aligned team, enabling team, platform team and complicated-subsystem team. Stream-aligned team deliver value for the client and end user. Enabling team helps the stream-aligned team overcome problems and detect missing capability in the system. A platform team is a grouping of the other teams that help accelerate the delivery of the product. The complicated-subsystem team is there to help when complicated problems arise that require specific expertise.

In Team Topologies they also describe three ways teams can interact: collaboration, X-as-a-Service and facilitating. Collaboration is about the teams working together. X-as-a-Service is when one team provides, and the other team uses it as a service. This might be something like one team having a database that they have setup and collect large amounts of data for, and another team then needs to use that data in the project they are working on. Facilitation is when one team helps and guides another team.

Q3. Things are best done centrally in cases where consistency and coordination are key and for decisions that are difficult to reverse. Infrastructure decisions like choosing a database benefit from being made centrally because they affect the entire organization and need to be consistent across multiple teams.

Centralizing decisions helps to avoid duplication of effort, ensure a unified approach and make the overall system more manageable. When decisions are made centrally teams can focus on their specific goals without having to reinvent the wheel for things like security or deployment strategies. This also allows for better scaling, as everyone can rely on a solid, shared foundation that is maintained at the central level.

Q4. In the 2000s standard practice was quite slow and had an analysis and design team, development team, test team, release team and operations team. Now teams deliver faster and can make changes fast based on feedback and consists of one development/operation team. The ABCDE of modern architecture is: Aligning, value, people and technology, Better value sooner safer happier, Continuous conversational governance, DevOps at enterprise scale and Evolutionary enterprise architecture.

Simon Rohrer's approach is practical and realistic, and he suggests that modern architecture should focus on outcomes rather than outputs. He argues that the architecture to evolve with business processes instead of outdated governance practices. He also advocates for supporting teams and avoiding review boards that can slow down teams and development. Instead following a community-driven approach can help teams deliver business value.

Q5. The continuous conversation is about having a constant dialogue between developers, business teams and operation. This enables faster feedback loops and ensures that development efforts are always aligned with business goals. Rohrer argues that it helps the bank remain adaptable to changes and ensures that updates can be quickly implemented. Like continuous integration it is important to also constantly get feedback and keep the different teams working at Saxo communicating so that they are always on the same page.

Q6.

Pipes and filters: This integration pattern is about breaking down a task into different steps. If a message is coming on each step might do something to transform the message before sending it of to the next step. Use case: In a company they might have some customer sales data from a database that needs to be transformed and reformatted so that it can be analyzed and used to gain insights into customer behavior.

Request Reply: A message is sent to a recipient that then gives back a response. This process is synchronous so the sender will wait for a reply before continuing. Use case: A service needs to send out a bill to a customer. It will send a message to a service that can get the customer data from the database and then when the billing service gets a reply it can proceed with the next steps.

Publish Subscribe: A message is published to multiple subscribers. The sender won't know who they are sending the message to or how many. All subscribers receive the message by subscribing to the channel where it is sent. User case: In the stock market updates on stock prices are constantly sent out to different software and websites.

Point to Point: In this pattern a message gets sent to one receiver and is usually done with a message queue where the sender sends out the message and a receiver picks it up. Use case: If you have a web shop when an order is placed it will be sent to some sort of warehouse service that processes the message so that it can be sent on to be packaged.

File transfer: One system generates and writes data to a file which is later used by another system or service. It is an older pattern but can still be used when real time messaging isn't necessary. Use case: A payroll service creates a file with employee salary data, and it is then later used by a accounting service who add it to their financial records.

Q7. In messaging patterns, the focus is on the message and the flow of the message. Conversation patterns focus on the participants in the conversation. The state is important in a conversation pattern it tells you where you are in the pattern. The conversation pattern is good when talking about error

handling. One example of a conversation pattern is subscribe notify pattern. It focuses on how the pattern works overtime and how the subscriber receives multiple messages. The message pattern publish subscribe focuses on how the message is sent out to the subscribers. Some of the problems with conversation patterns is dealing with deadlocks and error states. They are also difficult to visualize.

Q8. The core requirements for patterns start with the setting up the conversation or message. Then you some participants to talk to. This can be a direct conversation or there can be an intermediary. Then the last step is implementing resource management, error handling or recovering from errors.

Q9. Strangler (named after the strangler fig tree) pattern application is supposed to help evolve legacy systems and components. These systems are often a big part of organizations software and can be difficult to maintain and integrate with newer software. This pattern works by slowly replacing parts of the older system with modern microservices. This way you slowly build up new features and slowly phase out the legacy system. Replacing the entire legacy system all at once can be a complicated thing and can take a long time and the businesses requirements might change during the process. Using the strangler pattern allows you to test and deploy smaller services and allows the legacy system to continue operating.

Q10. In the video he discusses three diagrams which are rules, structures and requirements. They each have business focused side and an IT focused side. Rules represent the constraints and guidelines that the architecture must adhere to. They may include business rules, policies, or standards that govern how systems and processes should function. Structure diagrams show the components and their relationships within the architecture. In solution architecture this might show how a specific application is structured with its modules. In enterprise architecture it could represent the high-level arrangement of processes or IT systems in the organization. Requirement diagrams show the functional and non-functional requirements for the architecture. In solution architecture the focus is the specific project requirements. For enterprise architecture the requirements focus on the broader goals of the organization.

Rules example: There could be rules for how a system must comply with GDPR regulation such encryption and access control.

Structures example: For an online web shop you might have a structure diagram that shows the different components like shopping cart, payment, inventory and database.

Requirements example: This could be requirements for a web shop like user stories the user needs to be able to create an account and buy product. It could also be things like response time and scalability.