System Integration
Ola 2
Distributed Architecture and PatternsZ

By Kristofer, Mads, Michael and Søren

Q1 - Differences between Enterprise and Solution Architecture..... 2
Q2 - What does “Team Topologies” describe in the book Team Topologies..... 2
Q3 - Explain the reason and benefits described by Stefan Tilkov.....3
Q4 - Persuasiveness of Simon Rohrer’s Arguments on Modern Architecture..... 4
Q5 - Continuous Conversation and its Benefits to Saxo Bank.....4
Q6..... 5
Q7..... 6
Q8..... 7
Q9..... 7
Q10..... 7

Q1 - Differences between Enterprise and Solution Architecture

“What are the main differences between Enterprise and Solution Architecture. Describe the different roles that these play in designing large scale Systems? You can use resources like the video Enterprise Architecture vs Solution Architecture to help you answer.”

Enterprise architecture is about planning a strategic initiative while solution architecture is about describing the process of orchestrating software engineering to address an organization's needs.

Enterprise architects are concerned with how they can reduce costs, eliminate redundancies in technology and processes, and prepare for, mitigate and manage the impact of change. To operate effectively, enterprise architects must have a solid understanding of the organizations they work with or for.

The technical skills of a solution architect typically include software engineering and design, DevOps, business analysis and increasingly, cloud architecture.

Solution architecture’s value to enterprise architecture becomes even more clear. Where an enterprise architect is concerned with the EA’s current state, and the strategy to reach the desired future-state, solutions architects act in that strategic direction.

An enterprise architect’s strategic focus is high and their technology focus is low; technology architects operate in the reverse; and solution architects bridge the two.

Q2 - What does “Team Topologies” describe in the book Team Topologies

“In his video called Practical Architecture what does Stefan Tilkov say about the role of teams in modern architecture. He refers to a book Team Topologies – what “team topologies” does the book describe? Do you agree from your own experience that the team is a core part of a successful project?”

He describes that the most important thing is to allow yourself to change your mind, and that you're gonna thank yourself if you do and hate yourself if you don’t.

Team Topologies is the leading approach to organizing business and technology teams for fast flow, providing a practical, step-by-step, adaptive model for organizational design and team interaction.

Team driving development, how do you set up your team and how do they collaborate.

Examples:
Fundamental topologies:

- Stream-aligned team.
- Enabling team.
- Platform team.
- Complicated-subsystem team.

Interaction modes:

- Collaboration.
- X-as-a-Service.
- Facilitating.

“With a team-first approach, the team’s responsibilities are matched to the cognitive load that the team can handle. [...] For effective delivery and operations of modern software systems, organizations should attempt to minimize intrinsic cognitive load and eliminate extraneous cognitive load altogether, leaving more space for germane cognitive load.”

- Mathew Skelton, Manuel Pais: Team Topologies.

“Building and running a software system can be achieved using only four team types. Other team types can be actively harmful to an organization.”

- Mathew Skelton, Manuel Pais: Team Topologies.

“Team interactions outside these three core interaction modes are wasteful and indicative of poorly chosen team responsibility boundaries and poorly understood team purposes.”

- Mathew Skelton, Manuel Pais: Team Topologies.

Great things about team topologies:

- Explicit team-first approach.
- Autonomy at the heart of value creation.
- Technology-agnostic.
- Long-lived teams instead of project thinking.
- Based on actual experience, research, collaboration.

Q3 - Explain the reason and benefits described by Stefan Tilkov.

“Also in this video Stefan Tilkov explains why some things are best done centrally (for example at 23 min 30 seconds). Why do you think he is saying this? What does he claim are the benefits?.”

He believes some of the statements made within the book are very bold, and that he is opposed to the way of thinking that this is the only good way to do team focused development and that each team is unique and therefore may find that a certain way of doing their work may work best for them. This is the opposite of the way the book describes as the book is very rigid in how it should be viewed and used, and that any outside method could be disruptive for any group implementing the work methods described within the book.

Q4 - Persuasiveness of Simon Rohrer's Arguments on Modern Architecture

"In the video Architecting for Outcomes Simon Rohrer describes old fashioned and modern enterprise architecture. He talks about the A B C D E of modern architecture to compare modern and legacy ways of working. Do you find his arguments persuasive and if so why?"

In the video, Simon Rohrer discusses how traditional enterprise architecture was mostly static and monolithic, creating silos and inflexibility within organizations. He contrasts this with the modern approach, which is more adaptive and customer-centered. Rohrer uses the **ABCDE** framework to outline key principles of modern architecture, emphasizing **Alignment, Better Value, Continuous Conversational Governance, DevOps and Evolutionary Enterprise Architecture**. These factors promote agility, faster delivery, and responsiveness to changing business needs, which are increasingly important in today's fast-paced environment.

We believe his arguments are quite persuasive, especially in the context of how modern organizations need to evolve to stay competitive. By emphasizing alignment and continuous delivery, Rohrer argues that companies can avoid bottlenecks and lengthy development cycles that hinder innovation. His focus on experimentation and a data-driven approach supports iterative improvement, which aligns with modern agile methodologies. Overall, the **ABCDE** framework makes a compelling case for moving away from legacy systems to more flexible, outcome-driven architectures that align with business objectives.

Q5 - Continuous Conversation and its Benefits to Saxo Bank

"In the same video he explains the concept of the "Continuous Conversation". In what ways does he say it benefits Saxo Bank? How does he connect the Continuous Conversation to the Devops Infinity Loop"

Simon Rohrer introduces the concept of the **Continuous Conversation** as an essential part of modern architecture, particularly in how it benefits Saxo Bank. He describes it as an ongoing, iterative dialogue between various stakeholders, including developers, operations, and business teams. This communication fosters alignment and enables the teams to respond swiftly to new challenges and business needs.

The Continuous Conversation ensures that everyone remains in sync with the evolving goals of the organization. For Saxo Bank, this means that developers and operations are constantly aware of the business objectives, customer feedback, and technical constraints, leading to quicker adaptations and more successful outcomes. Rohrer links this concept to the **DevOps Infinity Loop**, which symbolizes the continuous feedback and iterative nature of the DevOps process. By keeping this "conversation" active, the bank can ensure that both development and operations are working toward shared goals, promoting agility, efficiency, and faster delivery of value to customers.

Q6

In the presentation there are listed 5 different Message-Based Architecture patterns these are:

Remote Communication:

Messaging systems that can enable communication between different systems or applications. They handle the serialization and deserialization of objects into formats that are suitable for sending messages (eg. json, xml or binary) and often support streaming of data to support continuous communication.

An example is an e.commerce platform that has to handle customer orders, inventory and payment. These services will be deployed in different locations, and messaging facilitates the communication between them. When a user makes an order, an application from the user's side sends a message to the inventory and payment services to update stock and process payments remotely, using messaging protocols to serialize/deserialize the data.

Asynchronous Communication:

Asynchronous models are where the senders and receivers operate independently. Senders can submit messages without waiting for an immediate response, and receivers can process the messages at their own pace, improving system scalability and resilience by allowing independent processing.

An example of this is a tracking system that allows customers to submit support requests that are processed asynchronously. Customers don't need to wait immediately for their response. Tickets are queued and the support team can handle them at their own pace. This allows customer requests to be handled at varying time, and the support team can process tickets based on priority, availability or load.

Platform/language Integration

A message based architecture that is ideal for integrating systems developed using different programming languages. Messages will be using a common paradigm, protocol or interface, such as Kafka, RabbitMQ or ACtiveMQ.

It's not uncommon that microservices are written in different languages. If we take an example like a media company, the video processing service can be built in python, while the recommendation engine uses Java. The platform will have to use a message broker to enable these services to communicate, passing data between services without dealing with the intricacies of each platform's implementations.

Reliable Communication

Message persistence is an important feature that improves reliability. Messages can be stored, either temporarily or permanently for queues, ensuring they are not lost even if the system fails. This guarantees resilience and that messages are delivered.

This can be used in a banking system, where reliable message delivery is a requirement. Using a message broker with persistence, the bank can ensure that even if the system fails, messages stored can be delivered once they are back online and ensure no transactions are lost.

Throttling

In throttling the receivers can control the rate of consuming data from a message queue. This is important for preventing overload, where a system may be flooded with too many requests. By controlling the flow of data, the system ensures that receivers process requests at an optimal pace.

A video streaming service is a good example of throttling. A streaming service like youtube, twitch or netflix, the backend service is responsible for transcoding the videos. This can be overwhelming when you potentially have millions of users and hours of video. By using a message queue, the transcoding service can pull messages at its own pace, ensuring that it processes videos at the pace capacity allows it to.

Q7

A messaging architecture uses the exchange of data between different systems by using messages. It often employs a publish-subscribe pattern, where messages are sent, and receivers or listeners handle them upon arrival. These systems can exchange individual messages or streams, but there is no built-in memory of previous interactions between systems. The primary focus of messaging patterns is on dispatching and delivering messages rather than controlling the flow or managing the relationships between messages over time.

A conversation architecture focuses on maintaining the context and the flow of a series of messages between the two systems. This can create an overly complex system when the messages have a long sequence of interactions to go through.

Gregor Hohpe notes several challenges with message based architecture.

- Completeness: Does the conversation actually complete.
- Deadlock: when two systems are not able to agree and are waiting for the other system to do something.
- Error states: As messaging architecture systems can be complex without a rigid sequence, error handling can be complex as well.
- Visualization: It is difficult to represent the state space of messages. The complexity has also made it difficult to sketch. Gregor Hohpe mentions that uml 2 has attempted to do this with sequence diagrams but failed.

Q8

The core requirements for patterns to be useful include:

- Setup: Patterns must address and resolve specific forces (conflicting requirements, challenges, or constraints). These forces often include aspects like discovery, trust, performance, and system complexity.
- Trade-offs: Patterns inherently involve making trade-offs. A pattern should provide a solution that considers the costs and benefits of different approaches, helping designers choose the best path based on the constraints and specifications they face.
- Interconnectedness in a Pattern Language: Patterns should fit into a larger pattern language, where each pattern has relationships with others. This allows for navigating complex design spaces by following patterns that build upon one another. As systems will interact with each other, it is important that some sort of protocol or handshake to ensure communication.
- Clear Structure and Vocabulary: For a pattern to be useful, it needs to provide a clear name, explanation, and structure, helping developers understand and develop the solution. This shared vocabulary allows for better communication among architects and system designers.

Q9

Q10