<question>According to Derek Martin, software architecture is primarily about:

<variant>Decision-making and its future impact

<variant>Writing clean code

<variant>Building microservices

<variant>Creating documentation

<variant>Managing development teams

<question>What is the main goal when considering options and costs in software architecture?

<variant>Creating a flexible system that can evolve without high modification costs

<variant>Minimizing development time

<variant>Reducing server costs

<variant>Maximizing system performance

<variant>Implementing all possible features

<question>The presenter describes coupling and cohesion as:

<variant>The yin and yang of software design

<variant>Optional concepts in software design

<variant>Contradictory principles

<variant>Modern programming paradigms

<variant>Database design principles

<question>Coupling in software architecture refers to:

<variant>The degree of interdependence between software modules

<variant>The process of combining multiple services

<variant>The speed of system communication

<variant>The process of deploying software

<variant>The relationship between developers and architects

<question>High cohesion means:

<variant>Elements within a module are closely related and focused

<variant>Elements within a module are loosely related

<variant>Modules are highly dependent on each other

<variant>The system has many interconnected parts

<variant>The code is highly complex

<question>When defining logical boundaries, the presenter suggests that:

<variant>Different pieces of functionality shouldn't be intertwined with unrelated functionality

<variant>All functionality should be in one service

<variant>Boundaries should never change

<variant>Boundaries should only be defined after deployment

<variant>Every entity should exist in only one boundary

<question>According to the content, a "product" entity in a system:

<variant>Can exist in multiple logical boundaries with different contexts

<variant>Should always be in a single service

<variant>Must be managed by the catalog service only

<variant>Should be replicated across all services

<variant>Must be stored in a single database

<question>The presenter describes a system with high coupling as:

<variant>A "distributed turd pile"

<variant>Highly efficient

<variant>Easy to maintain

<variant>The ideal architecture

<variant>Perfect for small teams

<question>To reduce coupling between logical boundaries, the presenter recommends:

<variant>Implementing messaging patterns

<variant>Using direct RPC calls

<variant>Creating more dependencies

<variant>Using a single database

<variant>Avoiding all communication between services

<question>Messaging in software architecture primarily helps to:

<variant>Create loose coupling between components

<variant>Speed up system communication

<variant>Reduce system complexity

<variant>Eliminate the need for databases

<variant>Increase system security

<question>CQRS stands for:

<variant>Command Query Responsibility Segregation

<variant>Complex Query Response System

<variant>Centralized Query Resolution Service

<variant>Command Query Response System

<variant>Complex Query Resolution Strategy

<question>The core principle of CQRS is:

<variant>Separating reads from writes at the application service layer

<variant>Using multiple databases

<variant>Implementing event sourcing

<variant>Creating microservices

<variant>Using domain-driven design

<question>According to the presentation, CQRS:

<variant>Is a simple concept often overcomplicated

<variant>Requires event sourcing

<variant>Must use separate databases

<variant>Needs complex schemas

<variant>Only works with microservices

<question>When implementing CQRS, which is NOT required:

<variant>Event sourcing

<variant>Separate paths for reads and writes

<variant>Command handling

<variant>Query handling

<variant>Application service layer separation

<question>The presenter suggests that defining logical boundaries is:

<variant>Difficult but important, especially for existing systems

<variant>Optional in system design

<variant>Only necessary for large systems

<variant>Only relevant for microservices

<variant>A one-time activity

<question>In terms of communication between logical boundaries, the presenter advocates for:

<variant>Asynchronous messaging patterns

<variant>Synchronous communication only

<variant>Direct method calls

<variant>Shared databases

<variant>No communication between boundaries

<question>What is described as a "gateway to a lot of options"?

<variant>CQRS

<variant>Microservices

<variant>Domain-driven design

<variant>Event sourcing

<variant>Message queuing

<question>According to the content, software architecture should NOT focus on:

<variant>Building in "what-if" scenarios

<variant>Making early decisions

<variant>Defining boundaries

<variant>Reducing coupling

<variant>Increasing cohesion

<question>The presenter suggests that when starting with CQRS, you should:

<variant>Begin with simple read/write separation

<variant>Implement full event sourcing immediately

<variant>Start with separate databases

<variant>Use complex message buses

<variant>Implement all possible patterns

<question>What is described as particularly challenging when defining logical boundaries?

<variant>Understanding the domain

<variant>Writing the code

<variant>Setting up databases

<variant>Managing teams

<variant>Deploying services

<question>The presenter's view on system evolution suggests that:

<variant>Systems should be designed to evolve over time

<variant>Systems should never change

<variant>Changes should be avoided

<variant>Evolution is only necessary for large systems

<variant>Changes are too expensive to implement

<question>When discussing product boundaries, the presenter emphasizes:

<variant>Different contexts having different needs

<variant>Using a single product definition

<variant>Centralizing all product data

<variant>Avoiding product-related services

<variant>Using only one database for products

<question>The relationship between coupling and system maintenance is described as:

<variant>Higher coupling makes changes more difficult

<variant>Higher coupling leads to easier maintenance

<variant>Coupling has no effect on maintenance

<variant>Coupling only affects new features

<variant>Coupling is irrelevant to maintenance

<question>According to the presentation, messaging patterns:

<variant>Enable asynchronous communication between boundaries

<variant>Should always use synchronous communication

<variant>Are only useful for small systems

<variant>Require direct coupling

<variant>Are optional in modern systems

<question>The presenter's approach to architecture suggests that:

<variant>Initial costs can provide future flexibility

<variant>All decisions should be made upfront

<variant>Changes should be avoided

<variant>Flexibility isn't important

<variant>Cost doesn't matter

<question>When implementing boundaries in an existing system, the presenter recommends:

<variant>Gradual boundary implementation

<variant>Complete system rewrite

<variant>Avoiding any changes

<variant>Waiting for a new project

<variant>Ignoring boundaries completely

<question>The concept of logical boundaries primarily addresses:

<variant>Functional organization of system components

<variant>Physical server locations

<variant>Database design

<variant>Network topology

<variant>Team structure

<question>According to the content, successful software architecture requires:

<variant>Making informed decisions about trade-offs

<variant>Implementing all possible features

<variant>Avoiding all coupling

<variant>Using only new technologies

<variant>Maximum system complexity

<question>The presenter's view on CQRS implementation suggests:

<variant>It can be implemented incrementally

<variant>It must be all or nothing

<variant>It requires microservices

<variant>It needs multiple databases

<variant>It should be avoided

<question>The main benefit of proper architectural decisions is described as:

<variant>Future flexibility at reasonable cost

<variant>Immediate cost savings

<variant>Faster development

<variant>Simplified deployment

<variant>Reduced team size