DUMPSARENA

ISAQB Certified Professional for Software Architecture -Foundation Level

iSQI CPSA-FL

Version Demo

Total Demo Questions: 8

Total Premium Questions: 89

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

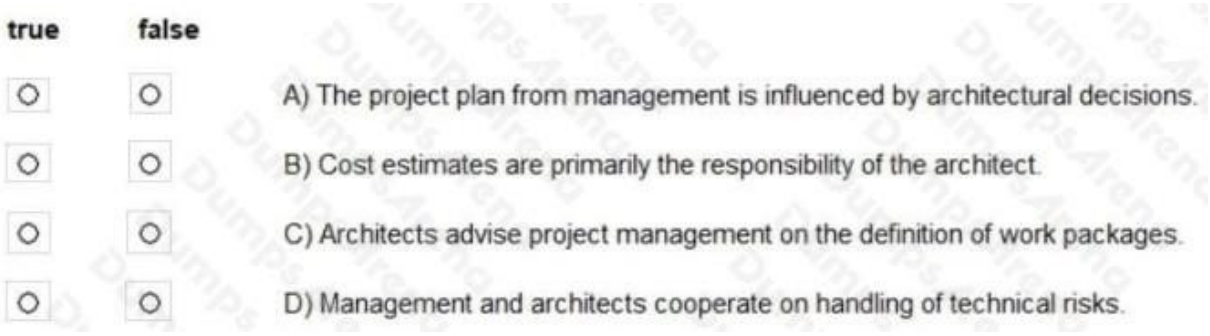
Topic Break Down

Topic	No. of Questions
Topic 1, Introduction and Motivation	2
Topic 2, Basic Concepts of Software Architecture	14
Topic 3, Design of Software Architecture	23
Topic 4, Quality	12
Topic 5, Documentation	29
Topic 6, Evaluation	5
Topic 7, Architectures and the Organization	4
Total	89

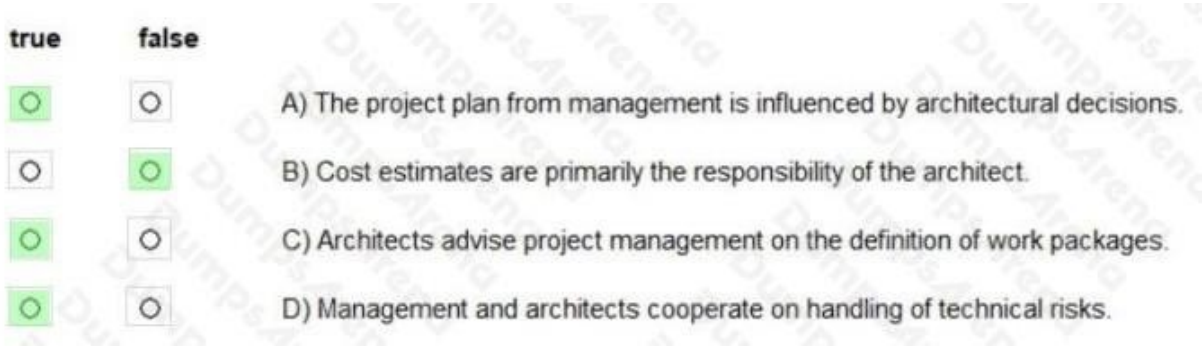
QUESTION NO: 1 - (HOTSPOT)

HOTSPOT

How does management and architects work together? Decide which statements are true and which are false. (Assign all answers.)
Hot Area:



ANSWER:



Explanation:

Management and software architects work together because architectural work is not isolated from project planning and control. The project plan is influenced by architectural decisions: selecting technologies, defining major building blocks, resolving technical dependencies, and addressing quality requirements can affect effort, sequencing, milestones, staffing needs, and delivery risks. Consequently, architecture provides essential input to management's planning activities.

Cost estimation is a management responsibility, although architects contribute important technical information. An architect can identify complexity, architectural risks, integration work, required technical skills, and likely consequences of quality requirements. Management brings these inputs together with budgets, schedules, organizational constraints, and commercial considerations to create and own the overall estimate. This division supports accountable project governance while ensuring estimates are technically informed.

Architects should advise project management on defining work packages. A useful work-package structure must take account of architectural components, interfaces, cross-cutting concerns, technical dependencies, and incremental integration. Architectural guidance helps ensure that the planned decomposition supports coherent implementation and avoids treating tightly coupled technical work as unrelated tasks.

Management and architects also cooperate in handling technical risks. Architects recognize and assess risks arising from technology choices, design assumptions, interfaces, performance, security, maintainability, and feasibility. Management incorporates those risks into project-level risk management, prioritizes mitigation work, assigns resources, and monitors the resulting actions. This collaboration connects technical risk insight with decisions about scope, schedule, and investment. The iSAQB CPSA-F curriculum overview describes the foundation-level focus on architecture work in its organizational and project context, while the Project Management Institute's discussion of technical project risk illustrates why technical expertise and project management need coordinated risk handling.

QUESTION NO: 2 - (SIMULATION)

Which characteristics of a building block are only visible in the whitebox view, and for which characteristics does the blackbox view suffice? (Assign all answers.)

ANSWER: See the explanation for the answer

Explanation:

Assign the characteristics as follows:

Whitebox view only:
D — Code structure of the building block
E — Algorithms used in the building block
G — Implementation details for the security requirements of the building block
Blackbox view suffices:
A — Public interfaces of the building block
B — Test coverage based on unit tests for sub-building blocks contained in the building block
C — Test coverage based on integration tests
F — Security requirements of the building block

A blackbox view describes externally observable properties: provided interfaces, required qualities or security requirements, and test-related evidence. A whitebox view is needed where the internal realization must be shown, such as internal code organization, algorithms, and the concrete implementation of security mechanisms.

QUESTION NO: 3

A client uses a service through an interface. The service implementation is replaced, but the interface contract remains unchanged.

Which quality characteristic is most directly supported for the client?

A. Modifiability

B. Usability

C. Performance efficiency

D. Confidentiality

ANSWER: A

Explanation:

Modifiability is most directly supported because the implementation can change without requiring corresponding changes in the client, provided that the published interface contract remains stable. The interface acts as an abstraction boundary between client and implementation.

This separation reduces the impact of implementation changes and supports independent evolution of building blocks. It does not by itself guarantee performance, security, or usability. See arc42 Building Block View for black-box descriptions based on responsibilities and interfaces.

QUESTION NO: 4 - (SIMULATION)

You are supposed to choose a software-architecture modeling tool for a software-development project. You create a suitable criteria catalogue for the choice of appropriate tools.

Which of the following factors can play a role in this? (Assign all answers.)

ANSWER: See the explanation for the answer

Explanation:

Answer: Select all options A through G.

A — Multi-user capability: relevant for collaboration and concurrent model work.
B — UML 2.x and SysML support: relevant because supported modeling notations must fit the project.
C — Document generation: relevant for producing and maintaining architecture documentation.
D — Model transformations for code generation: relevant where model-driven development is intended.
E — Code generation support: can be an important productivity and consistency criterion.
F — Compliance with standards: relevant for interoperability, organizational requirements, and longevity.
G — Purchase and licensing costs: relevant to tool budget and total cost of ownership.

A criteria catalogue for an architecture-modeling tool can include collaboration, notation support, documentation and generation capabilities, standards conformance, and commercial/licensing aspects. Therefore, every listed factor can play a role.

QUESTION NO: 5

Which statements about prototypes are correct in architectural work? (Choose three.)

A. They can reduce technical uncertainty.

B. They can validate selected quality requirements.

C. They can provide early feedback on a solution approach.

D. They must always be used unchanged as the production implementation.

E. They eliminate the need for architectural documentation.

ANSWER: A B C

Explanation:

A prototype can **reduce technical uncertainty** by testing whether an unfamiliar technology or integration is feasible. It can **validate selected quality requirements**, for example throughput, latency, or resource consumption under representative conditions. It can also **provide early feedback** on a proposed solution before substantial implementation effort is committed.

A prototype is usually focused on answering specific questions. It is not necessarily production-ready and should not automatically become the system's final implementation without deliberate evaluation and necessary hardening. Iterative validation is a central means of managing architectural risks.

QUESTION NO: 6

A building block shall be documented as a black box so that other teams can use it without knowledge of its internal structure.

Which information is most important in its black-box description?

A. Its provided and required interfaces

B. Its internal algorithms

C. Its local variables

D. Its source-code formatting rules

ANSWER: A

Explanation:

Provided and required interfaces are most important in a black-box description. They define the visible contract of the building block: the services it offers to clients and the services it needs from other building blocks. Together with

responsibilities, these interfaces enable teams to understand and use the building block without depending on its internal implementation.

Internal classes, algorithms, and implementation-specific data structures belong to a white-box description when they are architecturally relevant. The black-box perspective deliberately protects encapsulation and supports independent evolution of building blocks. See the arc42 Building Block View guidance.

QUESTION NO: 7

Which elements should be defined in the white-box view of a software building block 'foo'? Select the three most important elements. (Choose three.).

A. The dependencies of the internal building blocks of 'foo'

B. The legal contracts with the suppliers of the internal building blocks of 'foo'

C. The algorithms of the internal building blocks of 'foo'

D. The internal building blocks of 'foo'

E. The rationale for the decomposition of the building block

F. The sizes (in lines of code) of the internal building blocks of 'foo'

ANSWER: A D E

Explanation:

A white-box view documents the internal structure of a building block at an appropriate architectural level. Therefore, *The internal building blocks of 'foo'* must be defined: these are the constituent elements into which *foo* is decomposed. The white-box view must also describe *The dependencies of the internal building blocks of 'foo'*, because the relationships, communication paths, and required services between contained elements are necessary to understand how the overall building block collaborates internally.

The rationale for the decomposition of the building block is equally important. Architecture documentation should make significant design decisions comprehensible and traceable, recording why the chosen partitioning was made and what constraints or quality goals influenced it. This enables later maintainers and architects to assess whether the decomposition remains suitable when requirements change. These elements correspond to the arc42 building-block view, which uses white-box descriptions to show contained building blocks, their relationships, and relevant rationale. See the arc42 Building Block View guidance and the arc42 Architecture Decisions guidance.

QUESTION NO: 8 - (HOTSPOT)

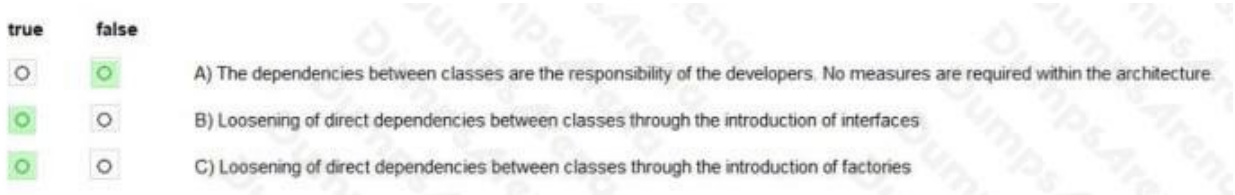
HOTSPOT

You are the software architect of a system that has run for many years and been extended repeatedly. An analysis of the source code has revealed a multitude of dependencies between the classes.

Which of the following measures are possible solutions? (Assign all answers.)
Hot Area:

true	false	
○	○	A) The dependencies between classes are the responsibility of the developers. No measures are required within the architecture.
○	○	B) Loosening of direct dependencies between classes through the introduction of interfaces
○	○	C) Loosening of direct dependencies between classes through the introduction of factories

ANSWER:



Explanation:

When source-code analysis reveals many dependencies between classes, an appropriate architectural goal is to reduce unnecessary direct coupling so that classes can change, be tested, and be reused with less impact on each other. Introducing interfaces is a suitable measure because a client can depend on an abstraction that expresses the required service rather than on a particular concrete implementation. This makes it possible to substitute implementations and localize implementation changes behind a stable contract. This is consistent with the Dependency Inversion Principle, described by Martin Fowler's discussion of dependency inversion, where high-level policy is protected from unnecessary dependency on concrete details.

Introducing factories is also a suitable measure. Object creation is often a source of direct dependencies because a class that explicitly constructs another concrete class must know that implementation and its construction details. A factory moves or encapsulates this creation decision, allowing the consuming class to request an object through an abstraction or creation service instead. The dependency is thereby redirected away from a directly instantiated concrete type, which supports more flexible configuration, replacement, and testing. The Factory Method pattern description explains how factory methods separate product creation from code that uses the product.

Accordingly, the correct selection recognizes both interfaces and factories as complementary refactoring and design measures: interfaces decouple usage from implementation, while factories decouple usage from concrete object creation. Together, they help manage the accumulated dependency structure of an extensively evolved system and establish clearer architectural boundaries.