Reference Model for Enterprise and Solution Architecture v18

Wednesday, 06 April 2022

This reference model has been written by Avancier Limited to support and enable training and examinations for certificates in enterprise and solution architecture.

It has been updated in a few places and is easier to read than the current BCS model

Contents

Introduction to the reference model

The goal of the reference model is to provide a controlled vocabulary for training and examinations in enterprise and solution architecture.

The reference model is designed to help:
- Examiners to scope and phrase examination questions.
- Examinees to understand terms and concepts in examination questions.
- Accredited Training Providers to scope training that leads to examinations.

Training Providers are expected to cover what the reference model contains.
They can add value to a training course, add more concepts and guidance, but additional content, outside the scope of the reference model, will not be examined.

For **examinations** the reference model should:
- provide the only source of terms and concepts used in Bloom level 2 examinations
- define the knowledge scope for practical questions in Bloom level 3 examinations
- help an examiner to ask fair questions with indisputable answers
- *limit* the examination questions that can be asked.

For **consistency and ease of understanding,** the reference model should:
- be internally consistent and coherent
- define not only concepts but also relationships between concepts
- be more an ontology (of concepts) than a dictionary (of words)
- be judiciously selective - not list all words architects use.

For **general and flexible use**, the reference model should:
- be broad, relevant to a wide range of architects' concerns
- be compatible with popular architecture frameworks, but not specific to one
- be compatible with popular modelling languages, but not specific to one
- not be tied to any one source - which may be updated separately.

Terms are included in this reference model to define examinable concepts and the context for them.
Beware that the wider IT industry uses some of these terms in ill-defined and sometimes contradictory ways, and different sources may use different terms for the same concept.

Acknowledgements
The reference model was built by Avancier Ltd over 20 years of research into architecture practices, standards and publications. It has been reviewed and approved for use in BCS examinations by X, Y and Z.

1 Systems, architectures and architects

“Enterprise architecture regards an enterprise as a system, or system of systems.” TOGAF
A business is a system where and in so far as its activities are particular and organized. Enterprise and solution architects design and plan changes to business systems in which human and computer actors play particular roles in particular processes that create or use particular business data (not ad hoc processes).

1.1 Encapsulation of systems

System
An entity that transforms inputs into outputs in an orderly way.
Its boundary is defined by the observer or designer who describes it.
It contains actors or components that interact to perform particular processes.

The generic metamodel of system elements below is adapted from the ArchiMate standard.

	Behaviours	Structures
External view	**Service contract:** a process defined as its requesters and consumers see it.	**Interface definition**: a declaration of available and accessible services.
Internal view	**Process:** a sequence of activities performed by one or more components.	**Component:** a subsystem capable of performing one or more behaviours.

Fig 1.1

Encapsulation
[A technique] that defines a system or component by its input/output interface.
The interface hides inner workings or processes from external entities.
It hides internal resources (notably data structures) from external entities.

Interface definition
[A structure] that declares what services a client can invoke.
It encapsulates one or components.
It may detail each service thus:
Logical signature: the service name, the inputs it consumes, and the outputs it produces.
Physical access: the address where the service can be found, and protocols used to invoke it.

System design principles
Principles for the specification and modularisation of a system include:

- external before internal,
- behavior before structure,
- business before technology,
- logical before physical,
- high cohesion within a component,
- loose-coupling between components.

Typically, finer-grained components are more tightly coupled, and coarser-grained components are more loosely coupled, so they can be managed and run independently.

1.2 Structural decomposition

System decomposition
[A technique] that divides a larger system into smaller components. You can see each component as a system in its own right and further subdivide it until you reach the level of atomic components you choose not to decompose.

Component
[An active structure] or subsystem capable of performing some required behaviour.
The required behavior can be defined in the form of one or more interface definitions.
A component can be replaced by any other component with the same interface(s).
EA frameworks address business, application and technology components.

Logical component
[A component] that specifies the capability one or more physical components.
It is abstract in the sense it is vendor and technology independent.
It may be specified by services provided and resources (notably data) needed.

Physical component
[A component] that is vendor-specific and/or can realise a logical component.

Dependency
[A coupling] between two components that means a change to the depended-on component requires impact analysis of the dependent component.

Granularity
[A measure] of the size of a structure or the duration of a behavior.

Modularity
Central issues in architectural design include

- The optimal granularity of components
- The optimal number of levels of component decomposition
- Avoiding unnecessary dependencies between components
- Avoiding unnecessary duplication between components
- Distributing components and integrating components

Data component
[A structure] in which some meaning or information is encoded.
It can be created and used, moved and modified.

1.3 Behavioral decomposition

Process decomposition
[A technique] that details an activity in a longer process as a shorter process of shorter activities. You can further subdivide it until reaching the level of atomic activities you choose not to decompose.

Process
[A behavior] a sequence of activities that produce a result of value, meet an aim.
Such as: deliver package, check credit, provide weather data, and consolidate reports.
It can be coarse-grained (build a house) or fine-grained (retrieve an address).
It may be defined externally in a service contact.
It may be defined internally as activities in sequence under logical control flow.
It may orchestrate subordinate processes.

Service
[A behavior] performed for a requester or consumer entity.
It supports or enables that entity by delivering one or more results of value.

Beware the term service is also used as a synonym for an interface definition, an application or an application component.

Service contract
[A behavior definition] that encapsulate a process by defining it in terms of entry conditions, exit conditions (aka results) and qualities, without reference what is needed to complete the service.
It may be documented along with other service contracts in a Service-Level Agreement (SLA) or an interface definition.

Service entry conditions
Event: the trigger of a behaviour, which may be a flow, time event, or state change.
Input flow: a material structure or message received by a process.
Precondition: a prior state that must exist if the process is to complete successfully.

Service exit conditions (aka results)
Output flow: a material structure or message produced by a successful process.
Post condition: a change of state resulting from a successful process; a change in the qualities of an object that is maintained, including any “added value”.
Value: the worth of an output or state change to an owner or consumer.

Service qualities
Measurable attributes of a service, such as: duration, cost, availability.

1.4 Abstraction techniques

To define architectures at a level that is more abstract than detailed design, architects use a mixture of abstraction techniques.

Delegation
[A technique] that simplifies clients by hiding work they delegate to servers.
A client component requests and/or consumes a service from a server.
A server component performs (realises, completes) services for clients.

Composition
[A technique] that hides small things ones inside larger ones.
A coarse-grained description features only large system elements.
A fine-grained description features more detail by way of smaller system elements.

Generalisation
[A technique] that simplifies by hiding differences between things.
A *generic* description is more widely applicable or universal.
A *specific* description is more narrowly applicable or unique; it may extend a more generic description with additional properties.

Idealisation
[A technique] that simplifies by hiding physical and/or vendor-specific details.
The classic idealisation hierarchy is conceptual, logical, physical, real.

Conceptual model
[A model] that defines terms and concepts in a business or problem domain without reference to any computer or software application.

Logical model
[A model] that excludes details of a system's physical implementation.
It is vendor-independent and portable.
It may specify services or processes to be performed, and/or data or abilities needed.
It leaves open the choice of particular components and technological products.

Physical model
[A model] that is vendor-specific or includes implementation details.
It specifies a particular entity that can be employed or deployed to realise a logical model.

1.5 Architecture levels

Enterprise
[A system] a business or organisation in which actors are directed to meet shared goals.
It may be a segment of an organisation, or a collaboration between organisations

Enterprise architecture
[An architecture] that gives a cross-organisational and strategic view of business systems.
It includes business, data, application and technology component portfolios.

Solution architecture
[An architecture] of a solution to a problem or requirement.
It is usually developed within the context of a programme or project.
It might be documented without reference to a higher or wider enterprise architecture.

Software architecture
[An architecture] showing the modularisation and integration of software components.
Some principles and patterns of software architecture are useful at higher levels.

The table below is a simple way to view the architect roles implied in this reference model.
Different organisations divide the work differently, and define other specialist architect roles.
Any one architect may work at more than level and in more than one domain.

Architect domain Architect level	Business architect	Data architect	Applications architect	Technology architect
Enterprise architect				
Solution architect				
Software architect / other technical specialist				

Fig. 2.1 The architecture work space

Strategic coordination view
[A description] that shows the degree to which an organisation aims to standardise and/or integrate business processes and business data, presented as two-by-two grid of process standardisation against integration.

High Process Integration	**Coordinated** (share data)	**Unified**
Low Process Integration	**Diversified**	**Replicated** (share processes)
	Low Process Standardisation	**High Process Standardisation**

Fig. 2.2 Strategic coordination view
(From “EA as Strategy” by Ross, Weill and Robertson).

1.6 Architecture domains

Architecture domain
A division or view of an architecture that addresses a broad set of concerns.
The four architecture domains below were established in the PRISM report (1986).
They are now the basis of countless EA frameworks, including TOGAF.

Business architecture
[A view] that identifies and relates business elements.
What the business delivers: business products and services.
How the business does it: business processes (scenarios, value streams).
What the business needs to do it: components and resources needed to perform processes.
Business elements may be mapped to goals and locations, to business data and applications.

Data architecture
[A view] that identifies and relates data elements:
- Data stores and flows used by business activities
- Data structures contained in the data stores and data flows
- Data qualities: data types, confidentiality, integrity and availability.

Data elements may be mapped to business activities and to applications.

Applications architecture
[A view] that identifies and relates application elements:
- Business applications needed to support business roles
- Data flows (messages) consumed and produced by applications
- Application use cases performed in the course of business activities.

Application elements may be mapped to business activities and to platform technologies.

Infrastructure/technology architecture
[A view] that identifies and relates technology components.
- Platform technologies and the services they offer to business applications
- Client and server nodes that applications are deployed on
- Protocols and networks by which nodes are connected.

Technology elements may be mapped to business applications, data stores and data flows.

Other domains
Other “architecture domains” such as motivation, security, governance, may span the four primary domains above.

1.7 Architecture capability – roles, goals and skills

Architect
[A role] that involves analysis of a context and requirements, analysing and choosing between options, defining an architecture, using principles and patterns, ensuring agreement of what is described, planning the move from the baseline state to the target state, and governing that change.

Higher level architect roles abstract from detail, operate more widely, with a broader scope, and may *govern* lower roles to a greater or lesser extent.

Enterprise architect role
[An architect] who takes a cross-organisational and strategic view of business systems, looking to.
- optimise business systems by digitisation, standardisation and integration
- exploit business data captured by business processes
- identify potential innovations in business processes
- maintain enterprise-wide business, data, application and technology component portfolios
- maintain cross-organisational and/or strategic road maps
- shape, steer and govern the work of solution architects.

Solution architect role
[An architect] who focuses on individual solutions and systems.
- addresses problems and requirements, related to specific processes and applications.
- aims to ensure the quality of solution delivery, in compliance with overarching goals, principles and standards where possible.
- describes solutions and govern their delivery, usually at a project level
- understands all domain/views well enough to work with all analysts and designers
- details a system architecture sufficiently for detailed design and building to proceed
- focuses on critical success factors, especially non-functional qualities.
- shapes, steers and governs the work of detail designers and implementers.

Enterprise architect goal
[An aim] Such as:
- Alignment of IS/IT to business strategies and goals
- Business agility and technical agility.
- Standardisation: of processes, data, applications and technologies.
- Integration: interoperation of processes, data, applications and technologies.
- Enablement of strategically beneficial change through long-term planning.
- Portability: supplier and technology independence.
- Simpler systems and systems management.
- Improved IS/IT procurement.
- Improved IS/IT cost-effectiveness.

Solution architect goal
[An aim] Such as:
- Support the goals (above) of enterprise architects
- Alignment of IS/IT to business processes and roles
- Quality of IS/IT project deliverables.
- Cost of IS/IT project deliverables (though a manager is usually *accountable*)
- Timeliness of IS/IT project deliverables (though a manager is usually *accountable*)
- Solution-level risk identification and mitigation.
- Application integration and data integrity.

- Conformance of solutions to non-functional and audit requirements.
- Conformance of solutions to principles, standards, legislation.
- Effective cooperation between managers and technicians.
- Governance of detailed design to architecture principles and standards.

Architect knowledge or skill

[A property] Such as:

- Holistic understanding of business and technical goals.
- Holistic understanding of business and technical environment
- Broad technical knowledge – including current trends.
- Broad methodology knowledge
- Analysis of requirements and problems
- Innovation.
- Leadership.
- Communication, political and soft skills (e.g. stakeholder management)
- Awareness of project management and commercial risks and issues.

2: Context and motivations (precursors)

The context for enterprise and solution architects includes business drivers, goals, principles, deadlines, benefits, costs, risks, system stakeholders and their concerns.

2.1 Foundation (not to be examined)

Architecture context
The context for architecture definition.
Including stakeholders, their concerns and other influencers.

Environment
The context in which a system operates.
The external entities that a system interacts with.
Cross-boundary inputs and outputs, events or flows.

External entity
[A component] that interacts with a system of interest.
Better named as a role rather than an individual acor.

Standard
[A concern] a widely-accepted definition of a structure.
It is intended to increase uniformity and interoperability of components or actors.

Best practice
[A concern] a widely-accepted definition of a process.
It is intended to increase uniformity and interoperability between processes.

2.2 Stakeholders and concerns

Stakeholder
[An actor or role] an individual, team, organization, or class thereof, having one or more concerns about or interests in a system, and/or power over the architecting of it.
Enterprise and solution architecture stakeholders include:

- Owners: business and IT board members, customers.
- Users: user representatives and domain experts.
- Managers: programme/project/change managers.
- Buyers: procurement/acquisition roles.
- Suppliers: service and product providers.
- Project team members: designers, builders, testers.
- Operators and maintainers, IT Services Management.

Stakeholder catalogue [An artifact] that lists stakeholders and associated facts.
E.g. concerns, interest level, power level.
Purposes: To help architects communicate with stakeholders and facilitate change.
To guide the development of architecture views.

Stakeholder management
[A technique] for ensuring concerns of stakeholders are understood and addressed.
An aim being to ensure that stakeholders support changes that are envisaged.
A stakeholder's position in a power/interest grid helps to prioritise attention to their concerns and determine a suitable communication plan.

Concern
An interest of stakeholders. E.g. system usage, cost, continuity, safety, modularity and all kinds of requirements and standards.
Concerns are generic (e.g. availability) whereas aims are specific (e.g. 24*7).
Architects identify concerns to select artifacts for presentation to stakeholders, and to ensure their interests are addressed and requirements are identified.

Sponsor
[A stakeholder] willing to apportion money or other resources to some work.

Request for architecture work
[A document] a request from a sponsor for an architect to architect one or more systems.
The first architecture deliverable to be recorded in an architecting process.

2.3 Requirements and constraints

Requirements catalogue
[A document] that lists requirement instances and their properties.
E.g. a reference number, description, source, owner, priority and requirement type.

Constraint (on work)
A factor that limits how far an organization can pursue an approach and/or meet a goal.
Common constraints include time, cost, resources and regulations.

Architecture requirement
A high-level requirement related to how business goals/objectives can be met.
Often, a change to a value stream, business scenario, or user experience.
Or the provision of management information.

Functional requirement
[A requirement type] related to services offered by a system.
It may refer to inputs and outputs, processes and business rules.

NFR: Non-functional requirement
[A requirement type] that quantifies how well, effectively or efficiently a system should deliver services. (See section 8.)

Explicit requirement
[A requirement type] that classifies requirements that are declared by stakeholders.

Implicit requirement
[A requirement type] that must be addressed, even if never mentioned by stakeholders.
Under any "best endeavours" obligation, the architect must be aware of the possibly implicit requirements below.

Audit requirement
[A requirement type] how an auditor can find the when/where/how/who of activities performed and data recorded, and replay events.
It may have implications for data records and retention.

Regulatory requirement
[A requirement type] including legislation and regulations that direct or constrain architecture work.

- Data protection: E.g. GDPR.
- Data freedom: E.g. Freedom of Information Act 2000.
- Disability and accessibility: E.g. UK Equality act. US Americans with Disability act. W3C Web Content Accessibility Guidelines.
- Shareholder protection and audit: E.g. US Sarbanes-Oxley act 2002. Basel I/II/III.
- Health and Safety.
- Intellectual property rights:

2.4 Scoping

Scope dimension
[A view] a dimension of scope.

- Breadth: scope of the enterprise, system or solution (see overview below).
- Domain in focus: business, data application or infrastructure technology.
- Depth: the detail to which description or design should be completed
- Constraints on work time, cost, resources etc.

Overview (breadth of enterprise or system)
[A view] that defines the breadth of a system from one or other perspective.

- Aim view: goal/objective decomposition or requirement catalogue.
- External view: a service portfolio or interface definition.
- Structural view: a context diagram, or decomposition thereof.
- Behavioural view: a value chain, process map or use case diagram.
- Data view: a conceptual, domain or business data model.

Context diagram
[An artifact] that shows a system's scope in terms of inputs consumed, outputs produced.
It shows the external entities (actors and/or roles) that send inputs and receive outputs.
The system is shown as a 'black box'.

Solution concept diagram
[An artifact] that shows an informal sketch or overview of a proposed solution.
It likely refers to aims and constraints on work to be done.
It likely indicates features and resources the solution will need.
Purposes: to engage stakeholders with how an envisioned solution will work and meet goals.

2.5 Other contextual terms

Contract
A declaration of the functional and non-functional parameters for one or more interactions between two parties where one is a client/requester and the other is a server/provider.

> *A service contract* declares the parameters any client uses to request one service.
> The service requester, performer and consumer may be different parties.
>
> *A collaboration agreement* covers all services one client can request from one server.

Measure
A measure of a system element (e.g. a component or process).
Usually a non-functional quality, requirement or attribute.
E.g. cost, size, duration, throughput or confidentiality.
It may be set as a target goal or requirement, or measured/reported in system operation.

Business case (before architecture)
[A document] that justifies work to build or change systems.
It will be outlined at the start and updated as need be.
It will be reviewed and refined several times while architecture work is done.
It may be decomposed into business cases for specific options or projects.
It should be completed when the full architecture and its implications are known.
(See the Migration Planning section for more on business cases.)

3 Architecture frameworks

3.1 Frameworks

A comprehensive architecture framework contains advice on processes for architecting, products for architecture definition, and the people involved.

Architecture Development Method (ADM)
[The architecture process] in TOGAF® which is centred on a cycle of 8 phases. A: Architecture Vision, B: Business Architecture, C: Information System Architecture (Data and Applications), D: Technology Architecture, E: Opportunities and Solutions, F: Migration Planning, G: Implementation Governance, and H: Architecture Change Management.

Architecture content framework
[A structure] for organising and storing architecture products, typically composed of deliverables, artifacts and entities.

Architecture deliverable
[An architecture] document, responding to a request for architecture work, requires approval.

> **Architecture or solution vision**
> The first response to a request for architecture work.
> It describes a target just enough to enable options to be compared.
> It may outline benefits, costs and risks, and work to follow.
>
> **Architecture definition or solution outline**
> [An architecture] that describes the high-level design of a target system or solution.
>
> **Detailed design**
> [An architecture] that describes the low-level design of a target system.
> Complete enough for building work to start.

Architecture artifact
A catalogue, matrix or diagram containing architecture entities

Architecture entity
An element that appears in one or more artifacts, such as process, organization, location, data entity, application and technology (POLDAT. The table below classifies some more entities.

	Required behaviours	Logical structures	Physical structures
Business	Business service Business process	Business function Role	Organisation unit Actor
Data	I/O data flow	Logical data model	Data store
Application	Application service/use case	User interface	Application component
Technology	Technology service	Technology interface	Technology component

Fig 2.3. Architecture entities

Architecture repository
[A data store] a database that stores architecture deliverables, artifacts and entities, and relates architecture entities as defined in a schema or architecture meta model.

Mappings between repository elements
[A correspondence] drawn between architectural entities in an architecture repository.
Mappings may be made the purposes of gap analysis, impact analysis, requirements traceability analysis and cluster analysis.

3.2 Architecture models

View
[A work product] that shows a part or slice of an architecture.
It addresses particular concerns.
It can be visual, graphical or textual. It may contain one or more models.
As an instance or example of a viewpoint, it conforms to the definition of that viewpoint.

Viewpoint
[A work product description] that typifies a view and provides a template for defining a view.
It defines the conventions for creating and using views to address concerns about a system.
It defines:

- what – the name of the viewpoint
- why - concern(s) that the viewpoint addresses
- who - stakeholder(s) who have the concerns
- how - model kind(s) used in the view.

Within one architecture, the viewpoint-to-view relationship is one-to-one.
However, one viewpoint may be used as a template for views of many different systems.

Model
[A work product] that simplifies or abstracts from a thing or another description.
It displays or records some properties of what is modelled.
It enables some questions to about it to be answered.
Architects build relatively abstract models of systems.

Model kind
[A work product description] that typifies a model and provides a template for building a model, such as may be found in modelling languages.

Modelling language
[A standard] that defines ways to represent architecture entities and the relationships between them, such as IDEF, UML and ArchiMate.

UML: Unified Modelling Language
[A modelling language] maintained by the Object Management Group.
Initially designed to help in OO software design, it is now used outside of that.
It includes structural models such as class diagrams and deployment diagrams.
It includes behavioural models such as use case, activity and sequence diagrams.

ArchiMate
[A modelling language] maintained by the Open Group.
Components, interfaces and services are representing in boxes using different symboles
It overlaps with UML, but is intended for more abstract architectural design.

MDE: Model-Driven Engineering
[A technique] used in methods and tools for forward engineering and reverse engineering, that is, for transforming a conceptual model to a logical model to a physical model, and/or the reverse of that process. (Such as in Model-Driven Architecture from the Object Management Group)

Reference model
[A pattern] a generic structure or classification used to create more specific models.
It can be a structure of components, processes or data elements.
It is sometimes applicable to a particular industry or business domain.
It can act as a design pattern.

3.3 Pre-defined classifications and reference models

Zachman framework

[A pattern] "A logical structure for classifying and organising the descriptive representations of an Enterprise that are significant to managers and to developers of Enterprise systems."

Zachman Framework v3		What	How	Where	Who	When	Why
Level	Stakeholder perspective	Inventory sets	Process flows	Distribution networks	Responsibility assignments	Timing cycles	Motivation intentions
Scope Contexts	Executive						
Business Concepts	Business manag''t						
System Logic	Architect						
Technology Physics	Engineer						
Tool components	Technician						
Operations Instance classes	Enterprise						

Fig. 3.1 The Zachman Framework

The 6 columns, though titled with interrogative questions, are mapped to architectural description facets or elements.

The 6 rows are primarily levels of realisation from context to operational systems.
But they are also mapped to stakeholder types and architecture domains.
Zachman says the rows should not be interpreted as levels of decomposition.

Enterprise continuum

[A pattern] a logical structure in TOGAF for classifying and organising architecture artifacts.
It can be drawn as a table or grid.
From top to bottom is ideal to real; from left to right is general to specific.

Enterprise Continuum	Foundation	Common systems	Industry	Organisation
	Universal building blocks for system construction	Used in most business domains	E.g. Telecoms or Banking	Your unique business
Context and requirements				
Architecture continuum				
Solution continuum				
Deployed solutions				

Fig. 3.2 The Enterprise Continuum

Reference model

[A pattern] a generic structure or classification used to create more specific models.
It can be a structure of components, processes or data elements.
It is sometimes applicable to a particular industry or business domain.
It can act as a design pattern.

4: Business architecture domain view

4.1 Business context

Mission
A statement that declares what an enterprise, business or organisation is about.
That is, its reasons for being; the essential products and services it offers.

Driver
An influence that shapes the directives and aims of a business.
Drivers are sometimes classified as Political, Economic, Social, Technical, Legal and Environmental (PESTLE), or as Strengths, Weaknesses, Opportunities and Threats (SWOT).

Vision
[An aim] that declares what an organisation wants to be or become.
An outline of an aspirational target state for an enterprise or business.
It may be defined in terms of measurable aims, or only a general direction to follow.

Directive
An influence or guideline, enduring and seldom amended, that steers or constrains behaviour or choices. Directives may be arranged in a hierarchical structure.

> **Principle**
> [A directive] that is strategic and not-directly-actionable.
> (Such as: Waste should be minimised. Data security is paramount.)
>
> **Policy**
> [A directive] that supports a principle.
> (Such as: The public have minimal access to business data. USB ports are disabled. Messages at security level 3 are encrypted.)
>
> **Business Rule**
> [A directive] that embeds a policy in data processing.
> (Such as: Access Level = Low if User Type = Public.)

Aim
A desired result or outcome declared or recognised by business system stakeholders.
It should be SMART (Specific, Measurable, Actionable, Realistic and Time-bound.).
Aims may be arranged in a hierarchical structure.

> **Goal**
> [An aim] that is strategic. It may be quantified using Key Goal Indicators.
> It may be decomposed into lower-level goals or objectives.
>
> **Objective**
> [An aim] that is more tactical than a goal.
> It should be quantified using Key Performance Indicators.
> It may be decomposed into lower-level objectives or seen as a high-level requirement.
>
> **Requirement**
> [An aim] a statement of need against which the outcomes of a system or work package can be tested.
> It should have acceptance tests and an acceptance authority.
> It may be captured in a requirements catalogue, service contract or use case.
> It should be traceable to higher level concerns, aims, directives or strategies.

4.2 Business structure concepts

Organisation units, roles, functions and capabilities may all be seen as structural business components, that group some actors or activities to meet some particular aims

Organisation unit
A division or department that gathers roles or actors into a manageable group.
It usually has goals/objectives, a manager, and a budget.

Organisation structure
The structure of units under which human actors are employed.
It is usually hierarchical, and shows the reporting line from a bottom-level unit to directors.

Role
A logical business component that groups activities performable by one or more actors.
Such as: loan applicant, expense claimant, expense claim approver.
It may define abilities required to play the role.

Actor
An entity able to play one or more roles.
It may be a person or organisation, an information system or technology.

Logical organization hierarchy
A common starting point for enterprise architecture.
A structure that successive divides a business into smaller business components
It typically stops at a 3rd or 4th level of decomposition.
It should be reasonably stable and contain no duplicate elements.
It structures a business regardless of who does what (actors) and how it's done (processes).
The structure is used to catalogue other architecture entities.
Heat-mapping may be applied to show where changes are needed, or to prioritise phases in a change program.

Function hierarchy
A logical organization structure composed of functions.
A business function groups lower-level functions or atomic activities that are cohesive, typically meet an aim
Core functions develop, market, sell and deliver business products and services.
Support functions are similar in different businesses, and candidates to be out-sourced.

Business capability hierarchy (or map)
A logical organization structure composed of capabilities.
A business capability groups lower-level capabilities or atomic abilities that are cohesive, typically meet an aim.
It may correspond to a function in a function hierarchy (such as sales), or to a stand-alone function (such as skill development).

Structured analysis
[A technique] that divides a business into logical functions or capabilities, which may be:

- mapped to organizations, roles or activities
- related by information and/or material flows
- sequenced in processes or value streams.

Structured analysis verification
[A technique] to ensure a business architecture is comprehensive and consistent.

It should be possible to map every *atomic* activity in a business process model to a higher-level business function or capability.
And to map every atomic business function or capability to a stage in a higher-level value stream, scenario or process.

4.3 Business behaviour concepts and artifacts

Business interface
[An interface definition] a collection of business behaviors accessible by an external entity.
It hides processes and resources needed to deliver the service.

Business service
[A service] that a business component is required to perform for an external entity
It is definable in a contract that encapsulates whatever process(es) are need to deliver the desired results.

Business process
[A process] performed by business actors in delivering a business service.
It may occur within one organisation or function, or coordinate activities in several.

Process diagram
[An artifact] that represents activities in a process from start to end.
The process may occur within a function or cross function boundaries.
Initially, architects focus on what is called the main, straight-thru, or happy path.
However, 80% of the complexity lies in the 20% of exception paths.

Value stream diagram
[A process diagram] that shows stages in an end-to-end business process and lists the activities in each stage. Functions, capabilities or resources may also be mapped to the stages.

Business scenario diagram
[A process diagram] that shows steps in an end-to-end business process, and maps them to the roles played by human and computer actors in each process step.

Process flow diagram
[A process diagram] that shows the control logic of activities in a process, and may include exception paths.

Service decomposition
A technique that divides longer services into shorter ones, such that one service is composed of dependent on subordinate services.

Process decomposition
A technique that divides longer processes into shorter ones, such that one activity in a longer process is composed of several lower-level activities.

Process automation hierarchy
[A technique] that decomposes a business process into steps, identifies application use cases needed at each step, and may go on to identify automated services required by a use case.

Lean
A collection of ideas for ensuring value-adding activities in the flow of a value stream run smoothly, quickly and without waste.
Those and related ideas (pull and perfection) were developed for application to processes for manufacturing hardware, and have been adapted for application to processes in human and computer activity systems.

4.4 Other business architecture artifacts and techniques

Organisation/actor catalogue
A list of the actors who pay roles in a enterprise, which associates the actors with organization units.
Purposes: to help architects identify and contact stakeholders with concerns or requirements. To define actors named in other artifacts.

Organisation/business function matrix
[An artifact] that maps organisation units to business functions, at whatever level of granularity suits its purpose.

Business communication diagram
[An artifact] that shows where business components interoperate.
Components interoperate by requesting and providing flows or services.
Each service delivers one or more results of value to the service requester or receiver.
The components and services can be modelled at any level of abstraction you choose.

SLA: Service Level Agreement
[A document] that records a business interface definition
A contract between a service provider and its customer(s).
It defines the legal context for delivery of services to the end consumers.
It lists services to be delivered, with performance levels as consumers see them.

Location catalogue
[An artifact] that lists the locations at which business activities are performed.
Sometimes represented graphically.

Business data model
See the data domain section.

Cluster analysis
[A technique] for gathering related items into a group.
It is used to group functions and processes, and used in exploratory data mining.

Affinity analysis
[A technique] that looks for relationships between services requested or activities performed.
It is used by retailers to perform market basket analysis.
This information can be used for purposes of cross-selling and up-selling, influencing sales promotions, loyalty programs, store design, and discount plans.

5 Data architecture domain view

A description (at logical and physical levels) of the major data structures an enterprise uses, their contents, and data management resources.

5.1 Foundation (not examinable)

Information
Meaning created and found by actors in a data structure.
The meanings can include descriptions, decisions, directions and opinions.

Data
Representations of information in a structure or medium of any kind.
It can be textual, numerical, graphical, pictorial, video, audio, biochemical etc.

Primitive data type
A generic data type: meta data that defines the properties of a variable.
Such as: String, Number, Boolean (holds a true/false value) and Null (absence of a value).
It defines the possible values for that type, the processes that can be performed on values of that type, the meaning of the data; and the way values of that type can be stored.

Variable
A data type used to hold domain-specific information.
Such as: Customer name, address and telephone number.
Variables are found in messages (inputs or parameters and outputs or returns) and memories (the fields of objects, and columns in database tables).

Data item
An instance of a data type that holds a specific data value.
Such as: an argument value, a return value, or a field value in a database.

Data structure
[A data type] a structure that arranges data items in one or more groups.

Structured data
Data that fits a pre-defined data type or data structure.
It often records real word entities or events.
It may contain references to unstructured data.

Unstructured data
Data containing text or images that do not fit a pre-defined data type or structure.
Such as: emails, voice and video.
However, it may contain recognisable structured items.

Meta data
[A description] of data, its structure, properties or qualities

Data dictionary
[An artifact] that catalogues data types and defines their meanings.
It may include business rules in the form of constraints on data values and derivation rules.
It may take the form of a canonical data model.

5.2 Data at rest (physical)

Data store
A container for persistent data structure, accessible by software.
Sometimes on discs, increasingly on solid state drives.

Data server
[A technology component] that hosts a database or file management system
It enables a data store to be accessed by applications.
It uses a particular physical data schema.

Physical data schema
[A data structure] in the format required by a particular data server.
It may realise a whole logical data model, or part of one.
It may be maintained by one or more application components.

OLTP data schema
[A data schema] optimised for transaction processing.
Typically, a normalised schema for a relational database.
Which enables direct access to any data entity instance, using its primary key.

Normalisation
[A technique] commonly applied to the data structure of an OLTP data store.
It stores each fact once, to ensure data integrity, and speed up data update processes.

OLAP data schema
[A data schema] optimised for the production of management information reports. Typically, a de-normalised schema for a data warehouse.

De-normalisation
[A technique] commonly applied to the data structure of an OLAP data store.
It duplicates some data storage to speed up data analysis and reporting processes

Document schema
[A data schema] optimised for the storage of forms and documents.
A document is stored in the structure it is created, and not modified thereafter.
It is typically an aggregate data structure XML or JSON format.
Changes to data values must be entered on new documents.

Big data schema
[A data store] characterised by a high volume of data, high velocity of data capture and a wide variety of data.

5.3 Data at rest (logical)

Data entity
[A data object] that represents a thing or event a user recognises as important in their domain of knowledge.
A data entity instance is identified using primary key composed of one or more attributes.
It can be related to other data entities in a larger data structure.
The data structure may be normalised, but does not have to be.

Data entity lifecycle diagram
[An artifact] that shows life of a data entity in terms of states it passes (through from creation to deletion) and the events that trigger state transitions.

Logical data model
[A data structure] composed of logically-related data entities stored and used by actors in business roles and processes.
It may define the data in one data store, or in several coordinated data stores.
It may be normalised, but does not have to be.

Data access path diagram
[An artifact] that shows the route that a process takes through a data model.
It is used to validate the data structure and study performance issues.

Business data model
[A data structure] composed of data entities recognised by business people.
It may be a very abstract model of data stored across a whole enterprise.
Or a more detailed model of data the data stores in one business domain.
Instead of a business data model, some enterprise data architects maintain simply business data entity catalogue, perhaps kernel entities only.

Data dissemination matrix
[An artifact] that maps data entities to data stores that contain them.
It shows duplication of data between data stores.
It is useful in analysis of change impacts, data mastering and security vulnerabilities.
It may be used to define, for a data entity, the master and copy versions.

Data entity / business function matrix
[An artifact] that maps data entities to the business functions that create and use them.
Cluster analysis can be used to cluster data that is created by the same functions, and functions that create the same data.

5.4 Data in motion (in flow)

Data flow (or message)
[A motion] a message, file, form, report, or display in which data passes from a sender to a receiver.

Data flow catalogue
[An artefact] that lists data flows, with attributes such as trigger, sender, receiver, transport technology and non-functional measures

Data flow structure
[A data object] conveyed in a data flow.

Logical data flow structure (or regular expression)
[A data flow structure] that is a hierarchy in which every element is part of a sequence, or an option of a selection or an occurrence of an iteration.

Physical data flow structure
[A data flow structure] represented in the format or schema required by a particular data format or technology.

Data format
[A standard] for the definition and organisation of a data flow structure.
Such as: Comma Separated Values (CSV), JSON, Extensible Mark Up Language (XML).

Data format standard
[A standard] for the content of a data structure.
Such as: EDIFACT, domain-specific XML Schema Definition (XSD).

Canonical data model
[A standard] that provides the “one true definition” of data types and structures used in data flow structures.
It defines what data can appear in messages between applications, and in the signatures of automated services.
It may be defined at a physical level using a data format standard such as XML.

5.5 Data qualities and integration

Data quality
[A property] of a data item, data structure or data store.
Meta data such as Confidentiality, Integrity and Availability (CIA).

Data integrity
[A property] that may embrace any or all of four qualities:

- Consistent: a data item (e.g. customer name) has the same value in every part of a distributed system, in all locations that data item is stored.
- Conformant: a data item obeys relevant business rules, sometimes in relation to another data item. Such as: an order must be for a known customer.
- Correct: a data item accurately represents a fact about an entity or event. The value of a data item is consistent with a fact in the real world.
- Controlled: a data flow has the same data content when it reaches its destination as it did when it left its source. OR data in a data store is not changed without authorisation.

Data disintegrity
[A property] an issue that may be redressed by one-off data quality improvement exercises, and by a variety of application integration patterns.

Master data management
[A technique] that enables an enterprise to maintain and/or find one “master” version of a data item or data structure, such as a customer or product data record.
It is supported by a range of application integration patterns and technologies, including some that hide the reality of disparate data sources from data consumers.

6 Software architecture domain view

6.1 Application modularity

This section contains selection of general terms and concepts used by software developers, for example in JavaScript.

Procedural language
A programming language in which programs are defined as procedures, and modules are defined in terms of the operations they can perform.

OO programming language
A programming language in which a program's modules are defined as classes at specification time and instantiated as objects at run time.

Application component
[A component] capable of performing automated behaviors.
It can be a whole application or a component within one.
It may be encapsulated behind an API, and may maintain some business data.

API
[An interface definition] a collection of automated behaviors accessible by software clients.
It identifies discrete services, may provide access to them, and hides what performs.

Delegation
[A process] whereby one component calls another to do the work.
Delegation usually implies invoking a component by passing it a message.

Stateful component
[An application component] that retains data in its local memory between invocations.
The state is lost if the component is removed.

Stateless component
[An application component] that does not retain data between invocations.
However, its transient state can be copied into a persistent data store.

Component-dependency diagram
[An artifact] that shows the design-time structure of a software application.
It shows which components depend on which other components

Sequence diagram
[An artifact] that shows how components cooperate at run-time to enable a process.
How a design-time structure behaves at run time is critical to meeting requirements.

6.2 Component integration

This section reflects a history of increasing distribution and integration of software systems, and the development of ways to connect loosely-coupled application components. However, there will always be use cases in which components are better closely coupled.

Local Procedure Call (LPC)
[A process] by which one component calls another component running on the same computer.
It is simpler, quicker and more secure than a remote procedure call.

Remote Procedure Call (RPC)
[A process] by which a process on one computer calls a process on another computer.
It is more complex, slower and less secure than a local procedure call.
The term usually implies a synchronous request-reply style of interoperation.

Distributed Objects (DO)
[A technique] for modular design and integration that employs an Object Request Broker.
An object request broker (ORB) is RPC-like middleware that enables the objects of an OO program to be distributed.
It may provide transaction management, security and other features.

Interface Definition Language (IDL)
A language for defining an API (not the procedures of operations/services in it).
It enables components coded in different languages and running on different operating systems to interoperate.
For example: the Web Service Description Language (WSDL) which includes a signature, protocol and web address for each operation/service.

Web Service
[A component] invoked over "the web" using an internet protocol and a published interface.
Initially, the term often implied the use of WSDL, XML and SOAP standards.
It is no longer related to any particular IDL, data format or internet protocol.

REST: Representational State Transfer
[A technique] for modular design and integration devised by Roy Fielding as a means to connect remote components using standard internet protocols.
It decouples distributed components so that client/sender components need minimal information about server/receiver components.
A RESTful client invokes a remotely accessible service using a domain name and an operation type available in an internet protocol, usually HTTP.
A REST-compliant server is identified by a domain name and offers only one service in response to each operation type in an internet protocol, usually HTTP.

OData
[An IDL] an evolution of REST that defines the data model of a remote web data store in an XML schema, and enables client application to invoke CRUD operations the entities in that data model.

EDA: Event-Driven Architecture
[A technique] for a modular design and integration that decouples the senders of news or update events from the receivers.
Any component can receive or read any event/message published by any other component.
It often implies using the publish and subscribe features of a middleware technology.
But can be implemented using a shared data space.

6.3 Component coupling

Synchronous
1: A request-reply style; a client must wait for a server to reply before continuing.
(The usual invocation from one COBOL module or Java object to another.)
2: A blocking style; a server serves one client at a time.
The caller and responder hold a channel open, blocking others from using it.
(The usual invocation style used by CORBA-compliant technologies.)

Asynchronous
1: A so-called fire-and-forget style in which a client does not wait for a server to reply.
(The usual style in email conversations.)
2: A non-blocking style in which a server can accept requests from several clients before responding to the first. (The usual style of Web Services.)
Typically, the server has a queue of incoming messages and releases the channel after a message is received.

Loose coupling factor
[A property] considered in deciding whether components should be tightly or loosely coupled.
The table below lists decoupling techniques.

Factor	Tight coupling	Decoupling techniques
Time	Synchronous request-reply Blocking servers	Asynchronous messaging Non-blocking servers
Location	Remember remote addresses	Use brokers/directories/facades
Naming	Clients use object identifiers One name space	Clients use domain names Multiple name spaces behind interfaces
Paradigm	Stateful objects/modules Reuse by OO inheritance Intelligent domain objects	Stateless objects/modules Reuse by delegation Intelligent process controllers
Data types	Complex data types	Simple data types
Version	Version dependency	Design to avoid version dependence Apply the open-closed principle
Protocol	Protocol dependency	Design for multiple protocols
Integrity constraints	ACID transactions	BASE: compensating transactions and eventual consistency

Fig. 6.3 Decoupling techniques

6.4 Component design patterns

Design pattern
A shape or structure of elements that commonly appears in solution design.
A tried and tested design that is tailored to address particular problems or requirements.

Façade
[A pattern] in which an interface that shields external entities from internal changes to a system or component.
It should reduce the coupling between client and server components.
It may aggregate fine-grained services into a coarse-grained interface.
It is usually stateless and does little or nothing but delegate work.

Model View Controller
[A pattern] that separates

- the processing of an input message (controller),
- the display of a user interface (view) and
- the retrieval and processing of persistent data (model).

Proxy
[A pattern] in which proxy components act as surrogates for remote components.
It is used in distribution of code between different name spaces, as in “Distributed Objects”

Observer
[A pattern] in which observer components monitor the state of a subject component.
A primitive kind of “Event-Driven Architecture”

Singleton
[A pattern] in which a component (or class) has only one instance (or object).

6.5 Communication patterns

Communication pattern
[A pattern] in which a client/sender application (or other actor) connects to a server/receiver application (or other actor).
Two broad communication styles, each subdivided into two narrower styles, are listed below.
There are other subcategories, not listed here.

Direct connection
[A pattern] in which clients/senders talk directly to servers/receivers.
There are two subcategories below.

> **Point-to-point connection**
> [A pattern] in which a message sent by one client/sender is received by one server/receiver.
> The client/sender knows the location of the receiver.
> The client knows what protocols and data formats the server/receiver understands.
> Strengths: simple and fast.
> Weaknesses: potential duplication of data transformation and routing code, reconfiguration costs on receiver address changes.
>
> **Direct broker connection**
> [A pattern] in which parties willing to communicate are registered (with end point locations) in a directory.
> When a client/sender wants to send a message to a server/receiver, the broker makes the introduction, and may establish client-side and server-side proxies.
> From then on, the parties talk directly or through proxies, as though using point-to-point connection.
> Not so simple and fast, but decouples clients/senders from server/receiver locations.

Indirect communication
[A pattern] in which clients/senders never talk directly to servers/receivers, they talk only through a mediator or shared resource.
There are two subcategories.

> **Indirect broker (mediated communication)**
> [A pattern] in which an indirect broker decouples communicating parties; it adds a layer of indirection between clients/senders and servers/receivers.
> This can enable communicating parties to work at different places and different times (asynchronously).
> It can shield one party from the effects of some changes to the other party.
> Mediator technologies include message brokers, message routers, message buses and publish-subscribe middleware.
>
> **Shared data space communication**
> [A pattern] in which parties communicate indirectly by reading and writing messages in a common data store, which might be shared memory, a message queue, a serial file or a database.
> Also known as: “space-based architecture” or “blackboard design pattern”.

7 Applications architecture domain view

Applications architecture depends on the lower-level design of fine-grained software components (section 6), but focuses more on whole applications, their use in business roles and processes, and how they are integrated.

7.1 Foundation (not examinable)

Application service
[A service] or operation that an application component is required to perform
Such as: log in, change password, submit insurance claim.
It supports and enables a business by capturing or retrieving business data.
It may be bundled with related application services into a portfolio or API.
It may be a use case at the human-computer interface, or a wholly automated behavior.

Application interface definition
[An interface definition] that includes services accessible by application clients.
It identifies services and hides what performs them.
It may be defined in a user interface or API.
It may be implemented as a facade component in its own right.

Application component
[A component] capable of performing automated behaviors.
It can be a whole application or a component within one.
It may be encapsulated behind an API, and may maintain some business data.

Microservice
[An application component] encapsulated behind an API, that is large enough to be regarded as an application, but small enough to suit agile development and enhancement.
It is usually a subdivision of what could be a larger application, and maintains a subset of what could be a larger database.
Data integrity is maintained using messaging passing rather than ACID transaction roll back.
So, using the BASE principle, designers must analyse the consequences of temporary data integrity and consider compensating transactions to restore it.

7.2 Application portfolio management

Application portfolio management
[A work process] to catalogue, classify, describe, and value the applications of an enterprise, with a view to rationalisation or optimisation of those applications.

Application portfolio catalogue
[An artifact] listing business applications and recording their properties.
It is usually structured so as to reflect the business function hierarchy.

Classification by architecture domain
[A pattern] dividing applications into one of three kinds:

> **Business application**
> [An application] that captures or provides data to support a business role or process. Such as: accounting; customer relationship management, patient administration.
> It has breadth in terms of use cases supported and depth in terms of software layers.
>
> **Generic application**
> [An application] that offers universal use cases. Such as: calculator, drawing tool, groupware, media player, spreadsheet, browser, word processor.
>
> **Platform application**
> [An application] technology component or "system software" that runs computer hardware or serves other applications. See section 9 for more detail.

Classification by value
[A pattern] that may measure both business value and technical value.
It may be presented in a two by two grid.

Applications communication diagram
[An artifact] that shows how applications are related by the exchange of data.
Typically, some kind of data flow diagram, or, where there are too many data flows, a dependency diagram.

7.3 Application behaviour

Use case diagram
[An artifact] that shows the uses cases supported by an application.
An application is scoped and logically defined by the use cases it supports.

Use case description
[An application service] at the human-computer interface.
A use case description defines a use of a system by an actor.
It is normally named as a goal in verb-noun form (e.g. assess claim).
It may be defined declaratively as a service.
It may be detailed in terms of a process with main and alternative paths.

Automated service
[An application service] that can be requested of a software component.
It is sometimes an ACID transaction.

Transaction
[An automated behavior] a unit of work, a buy-sell or client-server interaction between parties, e.g. between a user and a computer, or an application and a database.

ACID transaction
[A transaction] that is Atomic, Consistent, Isolated and Durable.
It can be rolled back if a specified precondition is violated.
Using a transaction manager to automate the roll back of a transaction preserves the integrity of stored data and simplifies design and development.
But is often impossible in loosely-coupled and distributed systems.

Compensating transaction
[A transaction] to handle the side effects of a process (or workflow) that started but could not complete successfully.
It may undo updates committed to databases, remove messages placed in message queues, send follow-up correction messages, or report cases of data disintegrity.

BASE (Basically Available, Soft State)
The opposite of ACID.
The principle used where interacting components are distributed and a process cannot be rolled back by a transaction manager.
The unit of work is workflow started by one component (basically available) and completed by another component.
If something goes wrong, since an update or output effect cannot be rolled back, compensating transactions may have to be designed.

Application/data entity matrix
[An artifact] that shows which applications create and use which data entities.
It may be completed with Create, Read, Update, and Delete entries.

Application interaction diagram
[An artifact] that shows how applications inter-communicate to enable a process.
It is often used to examine where time is spent in or between application processing steps.
Typically drawn as an interaction or sequence diagram.

7.4 Applications integration patterns

Applications can be integrated at different layers from UI down to database.
They can be integrated synchronously or asynchronously.
Architects must be familiar with integration patterns, and trade off their pros and cons.

User interface integration pattern
[A pattern] in which applications are integrated by moving data from one user interface and user interface to another (perhaps using RPA tools).)

RPA (Robotic Process Automation) tool
[A technology component] used to trigger or integrate applications at the user interface level.
In place of humans, robots complete tasks that involve entering data into applications.
Such as: receive an email containing an invoice, extract data from it, and enter that data into a book-keeping system.
It is a kin to a screen scraping for GUI testing tool, but sufficiently resilient, scalable and reliable for use in large enterprises.

On-line integration pattern
[A pattern] in which discrete operations or data stores are synchronised on-line, using either federated ACID transactions or compensating transactions (often using message/service bus tools).

Off-line integration pattern
[A pattern] in which discrete operations or data stores are synchronised off-line, often by overnight batch processes (often using ETL tools).

Data warehousing pattern
[A pattern] in which business data is copied from on-line data stores into a central database for reporting, often using ETL tools.
Data cleansing may be needed at any stage in the process.

Database consolidation pattern
[A pattern] in which baseline applications become user application components accessing one shared database.

Physical master data pattern
[A pattern] in which a common data entity is stored in a discrete database, where it can be accessed by any application with a pointer to the common data.

Virtual master data pattern
[A pattern] in which required data can be integrated at run time from several data stores or sources by some kind of broker application.
It features three layers of software components.

- User applications: present user interfaces, capture events from them and invoke broker applications.
- Broker applications: decouple by providing application services to user applications, and invoking data services from data app(s)
- Data applications: provide application services to put/get data to/from a particular database or other data source.

III-RM: Integrated Information Infrastructure Reference Model
The term used in TOGAF for the virtual master data pattern above.

7.5 Applications integration tools

Applications architecture is about integrating applications.
They can be integrated using a variety of tools, not just messaging.
Architects must be familiar with options, and trade off their pros and cons.
This section lists five kinds of integration tool.

ETL (Extract, Transform and Load) tool
[A technology component] that helps you to:

- *extract* data from data sources/senders,
- *transform* data items from one format to another, and
- *load* the reformatted data into data stores.

Useful for loading a data warehouse on a regular basis, loading a database during a one-off data migration, moving bulk data between databases.

Point-to-Point messaging tool
See “RPC” and “ORB” in section 9.

Web Service tool
See “Web Services” in section 9.

Message/service bus tool
[A technology component] that may:

- manage message queues
- store, route and forward messages between distributed components
- transform messages between protocols
- transform messages between data formats
- use a canonical data model in data format transformation
- manage federated/distributed transactions
- host procedures/workflows that orchestrate distributed components.
- support EDA using pub/sub mechanisms.

Using middleware can be more complex and slower than point-to-point integration, but has advantages where inter-component communication is one to many or many to one, and where the components at either endpoint are volatile.

7.5 Applications integration patterns (not to be examined)

Applications can be integrated at different layers from UI down to database.
They can be integrated synchronously or asynchronously.
Architects must be familiar with integration patterns, and trade off their pros and cons.

User interface integration pattern

[A pattern] in which applications are integrated by moving data from one user interface and user interface to another (perhaps using RPA tools).)

RPA (Robotic Process Automation) tool

[A technology component] used to trigger or integrate applications at the user interface level.
In place of humans, robots complete tasks that involve entering data into applications.
E.g. receive an email containing an invoice, extract data from it, and enter that data into a book-keeping system.
It is a kin to a screen scraping for GUI testing tool, but sufficiently resilient, scalable and reliable for use in large enterprises.

On-line integration pattern

[A pattern] in which discrete operations or data stores are synchronised on-line, using either federated ACID transactions or compensating transactions (often using message/service bus tools).

Off-line integration pattern

[A pattern] in which discrete operations or data stores are synchronised off-line, often by overnight batch processes (often using ETL tools).

Data warehousing pattern

[A pattern] in which business data is copied from on-line data stores into a central database for reporting, often using ETL tools.
Data cleansing may be needed at any stage in the process.

Database consolidation pattern

[A pattern] in which baseline applications become user application components accessing one shared database.

Physical master data pattern

[A pattern] in which a common data entity is stored in a discrete database, where it can be accessed by any application with a pointer to the common data.

Virtual master data pattern

[A pattern] in which required data can be integrated at run time from several data stores or sources by some kind of broker application.
It features three layers of software components.

- User applications: present user interfaces, capture events from them and invoke broker applications.
- Broker applications: decouple by providing application services to user applications, and invoking data services from data app(s)
- Data applications: provide application services to put/get data to/from a particular database or other data source.

III-RM: Integrated Information Infrastructure Reference Model

The term used in TOGAF for the virtual master data pattern above.

8 Design for qualities

8.1 Non-functional requirement types

Performance
[A requirement type] divided into two measures.
Throughput: volume or number of services performed in a time period.
Response or cycle time (aka latency): time taken from request to response or completion.

Availability
[A requirement type] the amount or percentage of time that the services of a system are ready for use, excluding planned and allowed down time.
Measures usually refer to availability at the primary site, excluding disasters.

Reliability
[A requirement type] the ability of a discrete component or service to run without failing.
Possible measures include mean time between failures (MTBF).

Recoverability
[A requirement type] the ability of a system to be restored to live operations after a disaster at the primary site.
Possible measures include mean time to repair (MTBR).

Integrity
[A requirement type] a term with four possible meanings defined under “Data Integrity”.
Possible measures include number of integrity errors reported in a time period.

Scalability
[A requirement type] the ability for a system to grow with increased workloads.
Possible measures include percentage increase in a time period.

Security
[A requirement type] the ability to prevent unauthorised access to a system.
Possible measures include number of security incidents in a time period.

Serviceability
[A requirement type] the ability to monitor and manage a system in operation.
Possible measures include time to detect issues and resolve them.

Usability
[A requirement type] the ability of actors to use a system with minimal effort.
Measures include Productivity, Learnability, Usability, Memorability and Error Rates.

Maintainability
[A requirement type] the ability to analyse, then correct or enhance a system.

Portability
[A requirement type] the ability to move a system from one platform to another, or convert it to run on another platform.

Interoperability
[A requirement type] the ability of entities to interact by exchanging information.
Logical interoperability requires common data types or translation between data types. Physical interoperability requires shared protocols and networks.

Extensibility
[A requirement type] the ability to add new features; a kind of maintainability.

8.2 Design for security

Design of human organisations to protect business systems from unauthorised or malicious access, to maintain the data qualities of confidentiality, availability and integrity.

Design for business (human and organisation) security
[A technique] defining anything that can be done outside of IT systems to secure business information, such as security guards, locks on doors, definition and roll out of policies and procedures.

Design for data security
Design to protect business data from unauthorised or malicious access, to maintain the qualities of data confidentiality, availability, and integrity.
Tom Peltier suggests rating the security level of a data item, data structure or data store as equal to the highest of the individual ratings (high, medium, low) awarded for Confidentiality, Integrity and Availability.

Information domain
[An entity] a uniquely identified set of items with a security policy that defines the rules that constrain access to data within the domain.

Identity
[A property] one or more data items (or attributes) that uniquely label an actor.
Such as: passport number or user name.

Encryption
[A process] to encode data items (in a data store or data flow) so that they are meaningless to any actor who cannot decode them.

Checksum
[A property] a redundant data item added to a message, the result of adding up the bits or bytes in the message and applying a formula.
This enables the receiver to detect if the message has been changed.
It protects against accidental data corruption, but does not guarantee data flow integrity, since it relies on the formula being known only to sender and receiver.

Digital signature
[A property] a cryptographic device that simulates the security properties of a handwritten signature.
More secure than a check sum, it is said to guarantee the data flow integrity of a message, since the signature is corrupted if the message content is changed.

Design for application security
Design to protect business applications from unauthorised or malicious use.

Identify management system
[An application] designed to ensure only authorised entities can access applications.
It ensures identification, authentication and authorisation of entities against permissions granted them.

Identification
[A process] via which an actor supplies their identity to an authority.
It is usually followed by authentication.

Authentication
[A process] to confirm that an actor is trusted - is the entity to which an identity was given.
It produces one of four results:

- true positive
- true negative
- false negative (which leads to wrongly-denied access), or
- false positive (which leads to unauthorised access).

It is usually followed by authorisation.

> **Three-factor authentication**
> [An authentication] that checks what users remember (e.g. password, mother's maiden name) and carry (e.g. credit card or key) and are (using biometric data).

Authorisation
[A process] for giving access to a trusted actor, based on that actor's known access rights.
It is usually followed by access.

Access
[A process] that locates data or processes of interest and retrieves them for use.

Design for technology security
What is done at the infrastructure technology level to protect business application systems from unauthorised or malicious access, to maintain the required data qualities of confidentiality, availability, and integrity.

IT service functions related security
- Identity & Access Management
- Continuity Management
- Security Intelligence
- Digital Forensics
- Security Analytics
- Audit, Network Monitoring
- Compliance Management
- Training & Awareness Programs, etc

Firewall
[A component] at the boundary of a network that can detect, filter out and report messages that are unauthorised and/or not from a trusted source.

DMZ: De-Militarised Zone
[A component] of a network, usually between the public internet and the enterprise network.
It uses firewalls to filters out messages that fail security checks.
It contains servers that respond to internet protocols like HTTP and FTP.

HTTPS
[A process] of normal HTTP interoperation over an encrypted Secure Sockets Layer (SSL) or Transport Layer Security (TLS) connection.
This ensures reasonable protection of data content if the data flow intercepted in transit.
(Aside: If an HTTPS URL does not specify a TCP port, the connection uses port 443.)

Web site security
[A process] whereby, for example, a web browser checks the public key certificate of a web server at the other end of an HTTPS connection.
The aims are to check the web server is authentic (who it claims to be) and that messages to/from the web server cannot be read by eavesdroppers.

8.3 Design for speed or response time

Removal of bottlenecks
The general advice is to look for bottlenecks and tackle them one by one.

Removal what slows things down
General techniques are to minimise or remove:

- distribution of processors and network hops
- database accesses, discs (replace by solid state drives)
- security overheads such data encryption.
- middleware.

Caching
[A technique] that copies persistent data nearer to the user than the original data source.
Generally good for response or cycle time; can raise data integrity and security concerns.

Database indexing
[A technique] that creates a list of pointers to stored data elements.
Some disable indexes during the day to optimise on-line update processes, and enable them at night optimise batch input and output processes.

Analysis of database access paths
[A technique] that studies the route of a process through a data store structure.
Since, a SQL programmer often does not know the access path their procedure takes through a database, it is advisable to use access path analysis and/or employ highly skilled SQL resources for critical database access programs.

Archiving of old data
Separation of update (OLTP) and reporting (OLAP) databases.

Throttling of throughput
To prevent servers being overloaded.

8.4 Design for throughput or capacity

Parallel processing
[A technique] to increase the amount of processing done in parallel.

Scale up
[A technique] that increases the power of one node.
Usually by adding resources, processors or memory.
Generally good for speed and throughput.

Scale out (aka clustering)
[A technique] that increases the number of parallel nodes, e.g. adding more nodes to a cluster.
Usually requires a load balancer to distribute service requests between nodes in a cluster.
Generally good for throughput; not always good for response time.

8.5 Design for availability (reliability and recoverability)

N+1 design
[A technique] to add an extra server to a cluster, to insure against failure.

Defensive design
[A technique] to
1. Design a client component so that it does not fall over if a server component does not work properly; typically using asynchronous invocation.
2. Design a server component so it does not depend on input data being valid. This means testing input data and preconditions before processing (the opposite of "Design by Contract" as promoted by Bertrand Meyer.)

Backing up
[A technique] to copy a data store for use if the original fails.
Or to restore individual files that have been deleted or corrupted.
Typically the last line of defence, and can be inconvenient to use.

Failing over
[A technique] that automatically switches over to a redundant or standby system, upon the failure or abnormal termination of the previously active system.
Failover happens suddenly and generally without warning.
Procedures must address "fail back" to normal operations.

8.6 Other techniques

Design for integrity
[A technique] such as reducing data replication, normalising stored data, switching on automated referential integrity checks, applying transaction management to update processes, removing caches, and consolidating distributed databases.

Design for serviceability
[A technique] to monitor servers, networks and instrument applications so that they report on what they are doing, and how well they are doing it. 6 Software architecture
Applications architecture (section 7) depends on the lower-level design of fine-grained software components discussed in this section. Ideally, enterprise and solution architects should be able to discuss detailed design with software architects

9 Technology architecture domain

A description (at logical and physical levels) of platform or infrastructure technology components and the services they provide to applications.

9.1 Foundation (not examinable)

Technology service
[A service] that a technology component is required to perform (e.g. start transaction, commit transaction, roll back transaction, send message, replicate data).
It supports and enables application components or other technology components by automating a generic/foundation infrastructure process.
It is commonly grouped with closely related technology services into a bundle (e.g. transaction processing services) that can be performed by a single technology component.

Technology interface definition
[An interface definition] that contains technology services accessible by applications.
It identifies services, may provide access to them, and hides what performs them.
It may be defined in the form of an API, or in a specific programming language.

Technology component
[A component] that is a unit of infrastructure, of platform software or hardware.
It may be encapsulated by the technology services it provides.
For example: operating system, device driver, web server, data server, message broker, data replication technology; also programming language and compiler.

Logical technology component
[A technology component] that is definable by services offered, independently of implementation and technology. For example, the specification of a database or message broker technology to support and enable applications. It is typically mapped to one physical technology component, but the relationship may be more complex.

Physical technology component
[A technology component] that is realised using a particular technology product, which may be hired or bought. It typically realises one logical technology component, but the relationship may be more complex.

Server node
[A technology component] that is a computational resource upon which components may be deployed for execution. Nodes can be connected a network structure or topology. Nodes may be virtual or physical. Nodes may be nested.

Communication network
[A structural view] of communication paths that enables technology components to send and receive data.

9.2 Enterprise technology rationalisation

Technology portfolio catalogue
[An artifact] that lists technology component types in a baseline or target architecture. It is usually arranged under the hierarchical structure of an enterprise technology classification.

TRM: Technical reference model
[A structure] for a technology component or services catalogue.
It can provide a requirement specification for technology rationalisation.
Such as: Client (user access) devices. Generic user applications. Operating systems. Database management. Middleware. Software development. Servers. Data storage. Networks. IT services management / operations. Environment. Security.

Technology rationalisation
[A technique] for studying the services provided by a baseline technology components and defining a de-duplicated target architecture. The basis of TOGAF versions 1 to 7.

1. Understand the baseline
    - Catalog baseline technology components (see TRM)
    - Catalog baseline technology services (see TRM)
2. Review the context and motivations
3. Design the target
    - Define target technology services
    - Define target technology components
4. Plan baseline-to-target migration
5. Govern delivery of the change.

Cloud computing
[An architecture] that features services provided to a customer over a WAN.
It is defined using contracts that hide what performs the remote services.
The service provider may pool the necessary resources.

Software as a Service
Provision of application services or use cases (e.g. order capture and payment validation) from a business application owned by a service provider; the customer owns only the data.

Platform as a Service
Provision of technology services (e.g. message delivery or transaction rollback) by infrastructure technology components owned by a service provider; the customer owns the business application as well as the data.

Infrastructure as a Service
Provision of basic computing technology services, processor speed and memory, by a remote service provider.

9.3 Solution technology definition

Device
[A node] with processing capability upon which components may be deployed for execution. Such as: application server, client workstation, mobile device, embedded device.
Devices can be nested.

Execution environment
[A node] on which components can be deployed in the form of executable components.
Such as: OS, workflow engine, database system, and J2EE container.
Execution environment instances are assigned to device instances.
Execution environments can be nested (e.g. database nested in an operating system).

Communication path
An association or channel between two nodes, through which they are able to exchange signals and messages.

Deployable component (or artifact)
A software deliverable (source file, script, executable, database table etc.) that can be deployed along with its descriptor to a node. One component can deployed to another.

Solution technology definition
[A technique] a process that starts with logical applications architecture and finishes with the technology architecture of the environment(s) needed to run the applications.
It should start by addressing show stoppers at the top and bottom of the technology stack.
Is the end user able and willing to use the client-end device?
Is data available from data servers when needed?
It may conclude with hardware and network diagrams.
It may proceed as follows:

1. Identify the context and requirements for the platform technologies
2. Establish baseline opportunities and constraints
3. Define client nodes and data server or source nodes (show stoppers)
4. Define intermediate web and app server nodes.
5. Map software components to nodes (with I/O protocols and connections)
6. Map virtual nodes to physical nodes
7. Define network(s) to connect the nodes
8. Refine to handle non-functional requirements
9. Define additional non-production environments
10. Govern deployment and transition from development into operations.

Software deployment diagram
[An artifact] that shows deployments of application and other software components to execution environments and nodes (virtual and/or physical).

Hardware configuration diagram
[An artifact] that shows the devices and indicates where they are connected by communication paths.

9.4 Communication networks and protocols

Network scopes
PAN: Personal Area Network
LAN: Local Area Network
MAN: Metropolitan Area Network
WAN: Wide Area Network
VPN: Virtual Private Network

Network layer
[A view] a level in a hierarchy of communication layers.
It may correspond to a specific platform technology.

Protocol
[A standard] for the process and rules used by message senders and receivers when they exchange messages in a telecommunication exchange.

TCP/IP 5 protocol layer stack
[A protocol stack] that is more commonly discussed than the old 7-layer OSI model:

> **Application layer:** such as FTP , HTTP, POP3, SMTP, RPC, SOAP and DNS.
>
> **Transport layer:** such as UDP (lightweight) and TCP (which ensures delivery or else times out).
>
> **Network layer:** handles the routing and forwarding of data at the packet level. IP is the dominant network layer protocol.
>
> **Data transport layer**: such as: Wi-Fi. Ethernet (a bus topology) and Token passing (a ring topology).
>
> **Physical layer:** the physical medium connecting nodes, such as: Modems, Optical fiber, Coaxial cable, Twisted pair.

IP (Internet Protocol)
[A protocol] a network layer protocol that sends data across a packet-switched internetwork, using IP addresses; the most prominent feature of the Internet.

IP address
[A property] logical identifies a node in a network that uses the Internet Protocol.
The first section identifies a local network; the second identifies a node on that network.

NAT: Network address translation
[A technique] that enables computers on a private network to access the internet using a single public IP address, by separating the address space within a LAN from the public internet.

Convergence (of telecommunication media)
The trend for one operating platform to supply many media, which enables equipment providers to combine data, voice and video services.

9.5 Connecting applications over networks

Process (computer sense)
An application or program instance, running on a computer. Each has a process number.
At run-time, one process can use several sockets to send and receive different kinds of I/O data.

Service Type
[A protocol] for a computer to send or receive one particular kind of I/O data (such as file, web pages or email).
One service type can be delivered via different ports.

NIC
[A component] a network interface card or network adapter that can connect a computer to a network.
At its simplest, one NIC is assigned one MAC address by the manufacturer at the factory, and is assigned one logical IP address by an engineer or the run-time environment.
The IP address on a NIC can be assigned many ports.

Port
[A component] that is assigned to one IP address for the purpose of sending or receiving I/O data using one service type.
The choice of port number can be made an engineer or by the run-time environment.
An international standard defines default port numbers for servers sending and receiving data via specific service types. Such as:

- An HTTP (unsecured) server listens for messages on port 80.
- An HTTPS (secured) server listens for messages on port 443.
- An SMTP server sends email using port 25.
- A POP3 server listens for email using port 110.

Socket
[A component] that holds data about the use of one port by one process.
It is identified by a process number and a port number (which has in turn been assigned to a logical network address and a service type).
Such as: an HTTP server listens for messages on port 80 at a particular IP address, and creates a socket for each process that sends a message.
Sockets are reused over time.

10 Migration planning

10.1 Migration planning concepts

Architecture state
[An architecture] at a point in time.
- A baseline architecture describes a system to be reviewed and/or revised.
- A target architecture describes a system to be created and implemented in the future.
- An intermediate or transition architecture defines a system between baseline and target.

Migration planning
[A work process] for turning architectures into a programme or project plan.
Architects should integrate the process into local programme/project management approaches such as MSP, PRINCE2 or PMI.

Gap analysis (baseline-target)
[A technique] to find items in one list or structure not in a comparable list or structure.
It is used in architecture frameworks to compare the elements of a baseline system with those of a target system, where each gap implies work to be done.

Migration path
[An artifact] a progressive series of architectures, each related to different state of an enterprise or system.

Architecture transition table
[An artifact] a table that shows when architectural entities are created, changed and removed through a series of transition states.

Work transition table
[An artifact] a table that shows when work pages start and stop through a series of transition states.

Roadmap
[A document]
1. A work plan that adds timescales to a migration path, so sits half-way between a migration path and detailed project plans.
2. A plan for how a resource (application or technology) will be updated, which may cut across several work plans.

Critical path analysis
[A technique] to construct a model of the project that includes:
- a list of activities required to complete a project (aka work breakdown structure)
- the duration of each activity
- the dependencies between the activities.

10.2 Business cases

Business case
[A document] a rationale and justification for spending time and money.
Generally speaking, the essential elements (defined separately) are ROI, Options, Impacts (work to be done and changes to be made) and Risks.
There should be a business case for work to describe an architecture, and then to implement an architecture as operational systems.
An outline business case is needed before architecture work starts in earnest.
It will be reviewed and refined several times, and perhaps decomposed into business cases for specific options or projects within the overall solution.

Return on Investment (ROI)
[An artifact] a statement of benefits gained minus costs spent – over a defined time period.
Costs must cover development, implementation, operation and maintenance.
Benefits may include money saved or regulations complied with.
E.g. the benefit of data integrity is to save the cost of data disintegrity.

Solution option
[An artifact] a design which can be compared with another, at any stage or level.
It may be presented using a business scenario.
The choice between options can be guided by the four techniques below.

Cost-benefit analysis
[A technique] to assess the costs and the benefits of a course of action and/or a proposed system.

Risk analysis (see section 10.1)

Gap analysis (options)
[A technique] for comparing two similar structures, to find items in one that are not in another.
It is used in a business cases to compare optional solutions.
It helps if the options are presented under the same structure as each other, or with reference to a more general structure.

Architecture trade-off analysis
[A technique] for comparing system options and trade offs between them with a view to selecting one option.
It may employ a technique such as a Pugh Matrx.

10.3 General planning concepts (not to be examined)

Course of action
[A work process] or plan that directs and focuses work to change a business to meet strategic goals and objectives and/or deliver the value proposition conveyed in a business model.

Programme
[A work process] a set of projects that are related by a common goal or shared budget, usually under one manager.

Project
[A work process] with defined start and finish criteria undertaken to create a product or service in accordance with specified resources and requirements.
Given a time span and a budget, it uses resources to deliver a required outcome, usually under one manager.

Work package
[A work process] a subset of a project's work breakdown structure, defined to yield defined deliverables.
May itself be decomposed into tasks assigned to different project roles.

Risk management
The identification, analysis, mitigation and containment of risks.

Risk analysis
[A technique] analysis of vulnerabilities that threaten the ability of a target system to meet requirements, especially non-functional requirements, including security.
Necessary before architecture starts in earnest, at several times in the process, and at several levels of design.

RAID catalogue
[An artifact] that lists risks, assumptions, issues and dependencies, which may be cross-referred to elements in requirements and/or solution documentation.
Cf. Risk Register in PRINCE2.

Risk
[An influence] a variation from what is expected or assumed.
It is usually a potential problem; an event that causes an issue if it occurs.

Assumption
[An influence] a belief or understanding that, if not true, could turn into a risk or issue.

Issue
[An influence] a problem that needs resolution.
It may be the realisation of a pre-identified risk.
It may be an assumption that turned out to be false.

Dependency (risk sense)
[An influence] a dependency upon an external actor or deliverable that is not under the management of the programme or project manager.

11: Architecture management

The organisation and processes that are needed to govern and implement an architecture description, both in development and in operation, including the management of changes.

11.1 Architecture implementation

Architecture implementation
[A work process] that realise an architecture through system development and deployment.
This requires programme and project management roles and processes.

Software Development Life Cycle (SDLC)
[A work process] centered on analysis, design software engineering, testing and roll out. There are agile, iterative and waterfall variants.

Waterfall
[A technique] a development process that places analysis, design, build, test and roll out in sequence. Engineers proceed from one kind of work to the next without significant iteration or parallelism between stages.

Iterative Development
[A technique] a development process that proceeds by increments, meaning that a working subset of the full solution is delivered as early as possible.
It is a foundation of the Unified Method and known as Incremental Development in DSDM.
It is an essential feature of agile methods, and may be used in non-agile projects also.

Agile Development
[A technique] a solution development process that is not only iterative, but also flexible about the requirements, the solution and the process being followed.
It favours

- negotiation over planning, and flexibility about requirements.
- early testing for usability and performance; user involvement and feedback is a prerequisite.
- short-cycle iterative development; it looks for the minimum change that adds value to a system, and strives to deliver that change in the next sprint/release.
- capitalising on the skills and knowledge of a small team.

Bimodal IT
Enterprise architecture and agile development.
Once a solution outline has been approved, agile development methods may be used.
Architects may govern the acceptability of changes during an agile project.

Transition
[A work process] that, once the architecture has been realised in the form of an operational system, hands over that system to the organisations that will operate it.

Transition into business operations
[A work process] that hands over a completed solution to a business unit for business process operation and management.

Transition into IT operations
[A work process] the hands over a production system to be run by a managed operations or ITSM organisation.

Transition into maintenance
[A work process] that hands over design and compile-time system to be maintained and perhaps enhanced by some kind of maintenance organisation.

11.2 Architecture governance

Governance
[A discipline] for monitoring and steering the management of an enterprise in accord with overarching drivers, goals/objectives and principles/policies.
It may be subdivided into:
- Corporate governance: the responsibility of the enterprise's executive board.
- IT governance: the responsibility of an IT board.
- Architecture governance: the responsibility of an architecture board.

Different enterprise relate these governance organisations in different ways.

Architecture governance
[Governance] of architecture, development and operations to ensure it conforms to pre-defined architectural requirements, principles, policies and models.

Architecture board
[An organisation unit] that maintains architecture principles and governance processes, promotes and ensures provision of architecture resources, and reviews compliance reports.

Architecture contract
[A document] that defines those architectural requirements, principles, policies and specifications that a system should conform to as it is built and when it runs.
Also defines any architecture stakeholder rights and interests that must be met.

Governing architect
[A work role] the architect who has been nominated by the governance organisation to ensure a system is built and/or run in accord with its architecture contract, to manage risk and to ensure the value of the system to its stakeholders.
The role may be played by a chief architect or design authority, architecture domain specialist or solution architect, depending on what is to be reviewed.

Architecture compliance review
[A work process] for monitoring work against architecture aims, directives and documents. Different reviews may be carried out at different points in system design and development.
Reviews may require a governing architect and/or use an architecture review checklists.

Architecture review checklist
[An artifact] a standard checklist of questions for an architecture compliance review.
The questions are general ones, not necessarily mentioned in the architecture contract.

Architecture conformance level
[A property] showing how well or how much of an architecture contract is met by a system, or an architecture is realised in a system.

Architecture compliance level
[A property] showing how well or how much of a system corresponds to its architecture contract and/or description.

Dispensation
[A document] a time-bound waiver from the terms of an architecture contract.
It is granted by a governing architect, and should be reviewed after the specified time.

Capability maturity model
[A reference model] for evaluating the maturity of an organisation and its processes.
The first was the capability maturity model (CMM) for software processes.
The capability maturity model integration (CMMI) widened CMM to cover other processes.
There are now various maturity models for architecture processes.

11.3 Architecture change management (not to be examined)

Architecture change management
[A work process] needed to manage changes to architectures, mostly stemming from changes to requirements or constraints, or operational systems. No different in principle from general change management (below).

Agile
[A property] willing and able to speedily respond to change.

Change management
[A technique] the roles and processes needed to both exercise **change control** to a baseline, and perform **configuration management.**

Change Control
[A technique] featuring the roles and processes needed within change management to monitor the potential sources of change, record change requests, perform impact analysis, deterrmine, make and release changes.

RFC: Request for Change
[An artifact] or form used to record details of a new requirement, a problem report or a change request to any configuration item.

Impact analysis
[A technique] to analyse the effects of a change, determine feasibility, update the business case and produce an impact analysis report.

Configuration management
[A technique] the roles and processes needed within change management to establish a baseline configuration and apply changes to that baseline configuration.
Involves work to:

- Identify and document the characteristics of each item.
- Define dependencies between items.
- Control the introduction of new versions of items.
- Report the status of configuration items and changes to them.

Baseline configuration
[A passive structure] a specification or product that has been formally reviewed and agreed upon.
The basis for further development.
Can be changed only through formal change management.
E.g. a contract, a requirements catalogue, architecture documentation, or a hardware configuration.

Configuration Item
[A data object] an item in a baseline configuration.
It could be a requirement, a source code component or a hardware device.
It can be at any level of granularity.
A “Component of an Infrastructure under the control of configuration management.
A configuration item can range from an entire system (hardware, software, documentation) to a single hardware component.” ITIL.

11.4 Architecture in operations [not to be examined]

The organisation and processes that are needed to manage the architecture description of an operational system.

IT service

[A service] provided an IT operations department. E.g.

- management of user roles and identities,
- client device configuration,
- storage administration,
- network provision, monitoring and analysis,
- server provision, monitoring and analysis,
- business activity monitoring,
- virtualisation,
- back up & restore,
- incident and problem management.

ITSM: IT Services Management

[A work process] the roles and processes for managing IT infrastructure and the services it provides.

IT4IT

A product of The Open Group that applies enterprise architecture principles to ITSM.
It defines a value chain with four primary value streams.
It decomposes each value stream into functions and defines artifacts that are produced by and exchanged between functions.

CMDB: Configuration Management Database

[A data store] a record of configuration item specifications including relationships among configuration items, where the items are significant to ITSM.

Asset management system

[A data store] a record of IT assets.
It is sometimes use to record end user devices, outside of the data centre.
It may be related to a CMDB.

Appendices: informative but not examinable

This section is not examinable; it is included in the reference model for the information of accredited training providers and maintainers of the reference model.

A taxonomy of architecture work

Architecting work element	**Work role:** a role in architecting.	**Architect:** one that can create, recall and use descriptions of things that it observes or envisages.
	Work process: an architecting procedure.	**Technique:** guides the performance of an activity to produce a work product. It may be associated with a pattern
	Work product: an output of architecting.	

A taxonomy of description concepts

Description: an abstraction that expresses a thing's properties by "intension".	**Design:** a description in the form of a plan (by drawing or other convention) of a buildable thing, artifact or system.	
	Model: a description that abstracts from a referent thing or more elaborate model; it can be a mental, documented, mechanical or other kind of model. It expresses some properties of the referent and enables questions to about it to be answered.	**Documented model (artifact):** a model in writing or drawing, which is stable and shareable.
		Mental model: that is recorded in a brain, less stable and shareable than a documented model.
	Type: a description comprising one or more property types embodied in an observed or envisaged thing, and encoded in a description or model of some kind.	**Property:** of a whole thing or element of a thing (a fact, form, function, feature, concept, condition, rule or attribute).
	Structure: a description that is organised, usually in one of the following ways: a list, a strict hierarchy (a one-to-many cascade with no duplicated items), a redundant hierarchy (with some duplication of items), a grid or matrix (relating items in two lists), or a network (items connected in many-to-many relationships).	**Pattern:** a structure that can be reused in similar situations to address similar issues, e.g. to organise components so as to cooperate to complete a higher level process.

Generic architecture concepts

There is an International Standard for "Systems and software engineering — Architecture description" (ISO/IEC/IEEE 42010:2011)

It takes no position on what a system is (see above) but does declare an architecture description shall *include* the following contents:

- identification and overview of the description
- stakeholders and their concerns
- an architecture view, composed of architecture models for each architecture viewpoint used
- a definition for each architecture viewpoint used
- correspondences and inconsistencies among the description's required contents
- rationales for architecture decisions made.

Note that this list of architecture description features is not a document template; the verb *include* allows for references between documents.

Relations between concepts

One model kind can be manifest in many models. E.g. a process flow chart notation is a model kind. A process flow chart of cooking an egg (showing activities under a control flow) is a model.

For one system, *one viewpoint is manifested in one and only one view.* E.g. in a process flow viewpoint:

- a concern is which activities are performed when;
- stakeholders include system actors;
- the model kind is a process flow chart notation.

One process flow view will contain *all* the process flow charts drawn for one system.

One view can contain several models of several kinds. E.g. in a business viewpoint:

- a concern is mapping processes to organisations and locations;
- stakeholders include managers;
- model kinds include a variety of diagram and grid types.

Any business view drawn to that viewpoint will contain a variety of diagrams and grids drawn for one system.

Principles followed in the reference model

The goal of the reference model is to provide a controlled vocabulary for training and examinations. This goal is supported by several principles.

1. The reference model entries do not say all that could be said, since they are intended to *limit* what examiners can set questions on and candidates have to learn. Trainers are free to explain any entry in depth if they wish to.
2. The reference model lists concepts to be understood, not all terms that might be used.
3. The reference model is not a general-purpose dictionary: it minimises synonyms (several words for one concept) and homonyms (several concepts for one word), since alternative terms and definitions undermine the goal of a controlled examination vocabulary.
4. reference model terms are chosen by compromising the principles of
    a. user warrant (what users are likely to use),
    b. literary warrant (what the literature says), and
    c. structural warrant (what helps to make the structure and content of the vocabulary clear and consistent).
5. The reference model does not invent terms other than is justified by the principle of structural warrant.
6. The reference model focuses more on discrete elementary concepts rather than aggregates of them (since the contents of aggregates are more disputable).
7. The reference model is not a standard; it describes architecture deliverables and techniques but does not mandate any of them or set any rules for what architects should do.
8. The reference model reflects the scope of current architecture frameworks; it does not speculate about future trends.
9. The reference model takes a best of breed approach (drawing, for example, from TOGAF, ArchiMate, ISO 42010, ITIL and PRINCE). There is minimal commentary on such sources – though there are a few remarks on where discrepancies may need to be recognised.

Change history

This reference model has been thoroughly refreshed since the previous version, with countless small changes too numerous to mention. The most notable changes are

1. Remove some peripheral terms and concepts. Remove the technology-specific addendum.
2. Refine the structure to balance sections, with a view to spreading exam questions more even across the sections.
3. Improve the consistency and coherence by introducing an overarching taxonomy and conceptual framework and map most entries to it.
4. Include non-examinable Foundation in each section.
5. Although the reference model must not recommend any particular standard or source; it should outline the core concepts in ISO 42010.
6. Section 1: Build up the general concepts and principles of systems.
7. Section 3: Include abstraction (formerly in section 1), and present the two architecture classification frameworks as tables.
8. Section 4: Clarify description of "structured analysis". Include capabilities, value streams and more artifacts
9. Section 6: Revise to put concepts in a clearer historical context. Add more on coupling v decoupling. Remove some detail on OO design patterns.
10. Section 7: Add Microservices and BASE. Strengthen sections on application communication patterns and integration patterns with a view to enabling more examination questions in this area.
11. Section 8: Design for Qualities (NFRs). Move content from other sections into this one

Trademarks