Reference Model for Enterprise and Solution Architecture v10.9.4

November 2017

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

How does it compare with the BCS version? It has been updated in a few places and is easier to read.

Introduction to the reference model (RM)

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

The RM 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 RM contains. They can add value to a training course, adding more by way of concepts and guidance, but any additional content, outside the scope of the RM, will not be examined.

For examinations the RM 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 RM 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 RM 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 particular source (which may be updated separately).

Many popular terms are used in ill-defined and sometimes contradictory ways. A term is primarily included in this RM for the purpose of labelling an examinable concept, rather than because the word is popularly used.

Acknowledgements
The RM was built from many years of research into architecture practices and numerous relevant publications. References (not required reading) are listed in the syllabus.

1: Architecture and architects

Both enterprise and solution architecture are about the design of business capabilities in which human and computer actors play roles in processes that create or use business data. This first section introduces basic ideas about architecture of this kind.

1.1: Foundation (rarely examined)

Architect: [a role] to design, plan and oversee the building of systems. It involves responding to requests for work, gathering contextual information, producing architecture descriptions and superintending lower-level design or construction.

Architecture description (aka **architecture):** [a work product] that describes the structures and behaviours of an operational system or capability. It may map system elements to motivations, constraints or work packages needed implement the system.

Domain layer	Behaviour Elements	Structure Elements
Business	Business services	Business functions, roles
Data and Applications	IS/App services	Business applications
Infrastructure Technology	Platform services	Platform technologies

Fig. 1.1 A table showing views of enterprise architecture description.

Architecting: [a work process] that creates an architecture description and a plan for building a system. It involves analysis of the context, engaging with stakeholders, making decisions, analysing and choosing between options, defining system elements, recording them in an architecture description and ensuring agreement of what is described. It often involves planning the move from the baseline state to a target state, and governing that change to ensure conformance of what is implemented to architecture descriptions.

Architecture framework: A comprehensive architecture framework contains advice on:

- Processes - for architecting
- Products - for architecture description
- People - architect roles.

1.2: Architectural design patterns (rarely examined)

Encapsulation [a technique] for defining a system or component by its input/output interface, by the discrete events it responds to and the services it can offer. It hides inner workings or processes from outsiders. It hides internal resources (e.g. data) from external entities, so the only way to access those resources is through the interface of the component.

Component-based design: [a technique] for defining a system in terms of interacting components (or building blocks). The components are defined by interfaces that list the services they provide, and the services (or server components) they require.

Design pattern: a shape that commonly appears in solution designs, a tried and tested design that is tailored to address particular problems or requirements.

Design pattern pair: a pair of contrasting patterns that each suit different situations. Architects choose between alternative patterns by trading off their pros and cons.

Hierarchical or centralisation pattern: [a design pattern] that centralises control in one place or component.
Anarchical or distribution pattern: [a design pattern] that distributes control to many places or components.

Hierarchical structure: [a design pattern] in which a node is divided recursively into subordinate nodes. Rules of thumb include: divide a node into about seven subordinate nodes, stop decomposition at three or four levels (or 1,000 atomic nodes).
Network structure: [a design pattern] in which a node can be connected to any other node.

Hub and spoke: [a design pattern] in which components communicate via a mediator: good where inter-component communication is many to many, endpoints are volatile; can be more complex and slower.
Point to point: [a design pattern] in which components talk to each other directly: good where inter-component communication is 1 to 1, endpoints are stable; can be faster and simpler.

Fork/Orchestration: [a design pattern] in which one component controls or schedules other components. It centralises intelligence about a process, a workflow that supervises and orchestrates the procedure.
Chain/Choreography: [a design pattern] in which components interact and cooperate. It distributes intelligence about a process. Each component does part of the work, then calls the next component (cf. pipe and filter).

Hierarchical layering: [a design pattern] in which components in a client layer delegate work to components in a server layer, and so abstract from that layer**.** This pattern is widely used to structure complicated systems (machines, networks, software applications and enterprise architecture). For example, the domains of business, application and infrastructure can be seen as client-server layers.
Peer-to-peer: [a design pattern] in which components can delegate to work to each other. Sometimes said to be a bad thing, a fragile and unstable structure, difficult to understand and maintain. Cyclic dependencies can undermine testability, parallel development, and reuse.

1.3: Architecture domains/layers as views

The four architecture domains below were established in the PRISM report (1986) and "EA Planning" (Stephen Spewak, 1993) and are now the basis of countless EA frameworks, including TOGAF.

Business architecture [a view] that identifies and relates business elements:
- business products and services
- business processes (scenarios, value streams) that maintain resources and deliver services
- business resources, roles and actors needed to perform processes.

Business elements may be mapped to goals and locations, to business data and applications.
EA is concerned with standardisation and integration of business roles and processes.

Data architecture [a view] that identifies and relates data elements:
- identifies data stores and flows created and used by business activities
- describes data stores and the data structures they contain
- describes data flows and the data structures they contain
- describes data qualities: data types, confidentiality, integrity and availability.

Data elements may be mapped to business activities and to applications.
EA is concerned with standardisation and integration of data between systems.

Applications architecture [a view] that identifies and relates application elements:
- identifies business applications and their uses to support business activities.
- describes coarse-grained applications rather than fine-grained software components.
- identifies application use cases
- identifies data flows consumed and produced by applications.

Application elements may be mapped to business activities and to platform applications.
EA is concerned with standardisation, integration and life cycles of business applications.

Infrastructure/platform architecture [a view] that identifies and relates platform elements
- identifies platform applications and the services they offer to business applications
- relates business applications to the platform applications they need
- defines the client and server nodes that applications are deployed on
- defines the protocols and networks by which nodes are connected.

Infrastructure elements may be mapped to business applications, data stores and data flows.
EA is concerned with standardisation, integration and life cycles of platform applications.

1.4 System elements

System elements can be divided into structure elements and behaviour elements

System: a bounded collection of interrelated elements. An activity system is a network of actors/components playing roles in processes to produce desired effects by maintaining system state and/or transforming inputs into outputs. It can be encapsulated behind an interface. It is open, meaning it interacts with entities and events in its environment.

Structural view: [a description] that shows what a system is made of. It shows actors/components and how they are connected in structures or accessed via interfaces.

Behavioural view: [a description] that shows what a system does. It shows services and/or processes that run from start to end.

	Behavioural view	Structural view
External view	**Event/Service** A trigger or encapsulation of the processing of a discrete event or service request	**Interface** A collection of services that are accessible.
Internal view	**Process** An activity sequence triggered by an event or service request, leading to a result	**Actor/Component** A subsystem that can performs processes

Fig. 1.4a A generic meta model of system elements.

System element: a unit of an operational system that can be described. Systems elements can be arranged in a grid as shown above. Since one person's system is another's subsystem, this model can be applied at any level of granularity.

Behaviour element: A thing that happens. A unit of activity triggered by an event and (providing its preconditions are met) terminating with the production of an output or other response.

> **Service:** A discrete stimulus-response behaviour, specified in a service contract that encapsulates process(es) needed to realise the service. A service contract is divisible into three main parts.
> - The signature: the service name, inputs and outputs.
> - The rules: the preconditions and post conditions of the service.
> - The non-functional characteristics: including performance and commercial conditions.
>
> **Process:** A sequence of activities, performed by one or more components, that delivers a service or result of value. There are human processes, computer processes and hybrid human-computer processes (sometimes called workflows or use cases). A process may be documented using a flow chart or interaction diagram.

Structural element: what a system is made of, a unit which is addressable in space**.**

> **Active structure element:** A component responsible for performing behaviours, which may be specified at logical or physical levels. A subsystem that can be related to others by requesting or delivering services. It can be replaced by any other component with the same interface(s).
>
> > **Logical component:** A structure element specified in terms of services provided and/or processes performed and abilities required. (Related terms: interface, function, capability, role.)
> >
> > > **Interface:** a collection of services accessible by clients; it identifies services, may provide access to them, and hides what performs them. (See footnote.)
>
> **Physical component:** A structure element that can be hired, bought or built to realise a logical component and perform required behaviours. (Related terms: component, organisation unit, actor.)

> **Passive structural element:** a structural element that is acted upon or in.
>
> > **Object:** an item or structure that is made, moved or modified. E.g. a data item, data structure, or any kind of structural component.
> >
> > **Location:** a place where actors/components and interfaces are found and work is done.

View Dpmain layer	Required behaviours	Logical structures	Physical structures
Business	Business service Business process	Business function Role	Organisation unit Actor
Data and Applications	Data flow IS/App service	Data entity Application interface	Data store Business application
Infrastructure Technology	Platform service	Platform interface	Platform application

Fig. 1.3b A conceptual framework for system elements.

Footnote: The term interface is used to mean other things, for which other terms may be favoured.

- The **signature** (name, inputs and outputs) of one service.
- A **data flow** between sender and receiver components.
- The **protocols** used to exchange data between components.
- The **channel** via which data flows are passed.

1.5: Architect responsibility level

Architect role [a work role] that may be classified by architecture level and/or domain.
Higher level roles abstract from detail and operate more widely, with a broader scope.
Higher level roles may *govern* lower roles to some extent, but may not directly manage them.
An architect may well work at more than level and in more than one domain.

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

Fig. 1.6 The architecture work space

Enterprise architecture [a responsibility level] that strives to:
- take a long-term, strategic view
- take an enterprise-wide view, treating the whole enterprise as a system
- address cross-organisational concerns and goals
- improve business processes by digitisation, standardisation and integration
- exploit business data captured by business processes
- identify potential innovations in business processes
- understand the enterprise's estate and maintain enterprise architecture collateral
- set out cross-organisational and/or strategic road maps and migration programmes.

Solution architecture [a responsibility level] that
- is relatively tactical: addresses specific problems and requirements, related to selected business processes and applications.
- aims to ensure the quality of a particular solution delivery, in compliance with overarching goals, principles and standards where possible.
- describe solutions and govern their delivery, usually at a project level
- understands all domain/views well enough to work with others: e.g. a general solution architect may work in partnership with a lead business analyst and/or software architect
- ensures the architecture of a system is sufficiently detailed for work to be planned and detailed design and building to proceed
- shapes projects and then govern others according to agreed principles and plans
- is responsible for delivery quality: focuses on critical success factors, especially non-functional qualities.

Software architecture [a responsibility level] that addresses the modularisation and integration of software components. It describes the fine-grained structure and behaviour of a business applications. It may follow principles and patterns for modularisation and integration.

Aside: The principles and patterns of software architecture (e.g. encapsulation, cohesion and decoupling) are useful at higher levels. Architects should be familiar with such principles and able to apply them appropriately to their context.

1.6: Architect goals and skills

This section is deliberately limited; these short lists are sufficient for examination purposes.

Enterprise architect goal [an aim] E.g.
- Alignment of IS/IT to business processes, roles 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: vendor and technology independence.
- Simpler systems and systems management.
- Improved IS/IT procurement.
- Improved IS/IT cost-effectiveness.

Solution architect goal [an aim] E.g.
- Support the goals of enterprise architects
- 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] E.g.
- 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.

1.7: Other architecture domains (rarely examined)

In addition to the four primary architecture domains, four other architecture domains are named below.

Information architecture: a term for the broad domain that includes structured data architecture, content management and knowledge management. EA frameworks (and this reference model) focus on structured data.

Information systems architecture: a term often used to mean the combination of applications architecture and data architecture. It depends on but does not include the technology platform.

Software architecture: the internal structure or modularisation of a software application. This is software architecture at the lowest level of granularity (as discussed for example in "Patterns of enterprise application architecture" by Martin Fowler). It is usually below the level of modularity that enterprise and solution architects define. However, there is no rigid dividing line

Security architecture: a term covering the various design features designed to protect a system from unauthorised access. Not a cohesive architecture on its own so much as features of the other architecture domains – business, data, applications and infrastructure.

2: Motivation and context

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

2.1: Foundation (rarely examined)

Architecture context: the setting of and influences upon a system. It contains all stakeholders, systems, concerns, influences and information that architects should be aware of. It includes, but is not limited to, what is understood of the wider environment.

Environment [a view] that describes operational context in which an enterprise, system or solution operates. The focus is on external entities that system elements interact with, and cross-boundary inputs and outputs, events or flows.

External entity [a component] outside the business or system of interest, which interacts with that business or system by requesting or supplying services. Defining external entities as logical *roles* makes a model more stable and flexible. Defining them as physical *actors* can make a model more immediately understandable.

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.
- Managers: programme/project/change managers.
- Buyers: procurement/acquisition roles.
- Suppliers: service and product providers.
- Designers, Builders, Testers, other project team members:
- Users: representatives and domain experts.
- Operators and Maintainers, IT Services Management.

Stakeholder catalogue [an artefact] that lists stakeholders and associated facts, such as concerns, interest level, power level, and plans for communicating with stakeholders.

Stakeholder management [a technique] for ensuring concerns of stakeholders are understood and addressed, with a view to ensuring 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 a stakeholder in a feature, influence on or requirement for a system. (E.g. system behaviour, development cost, disaster recovery, safety, all requirements, all standards, a particular standard, modularity.) A concern (e.g. availability) is more generic than a particular requirement (e.g. 24*7).

Sponsor [a stakeholder] who is 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: Influences

Influence: a belief, fact or statement that acts as a requirement for or constraint on behaviour or choices. Examples include instances of drivers, directives, aims (defined below).

Driver [an influence] a force, recognised by managers, that shapes the directives and aims of a business. Driver areas and types include Political, Economic, Social, Technical, Legal and Ecological (PESTLE) and Strengths, Weaknesses, Opportunities and Threats (SWOT).

Directive [an influence] a principle or policy, 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. (E.g. Waste should be minimised. Data security is paramount.)
>
> **Policy** [a directive] that supports a principle. (E.g. The public have minimal access to business data. USB ports are disabled. Messages at security level 3 are encrypted.)
>
> **Business Rule** [a directive] that implements a policy in data processing. (E.g. Access Level = Low if User Type = Public. See Process rule and Data rule.)

Aim [an influence] a goal or objective that is declared or recognised by business managers, or a requirement for a particular endeavour or system. 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 may support one or more higher-level goals. 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 with which compliance can be demonstrated in a specific solution or project. It should have acceptance tests and an acceptance authority. It may be captured in a requirements catalogue or in the text of a service contract or use case. It should be traceable to higher level concerns, aims, directives or strategies.

From the top-down, goals are decomposed into objectives. From the bottom-up, requirements are traced to objectives.

Footnote: Goals and objectives may appear in a “balanced score card” that spreads top-level aims across four categories, then cascades them down the organisation structure, decomposing them at each level.

2.4: Requirement types

Requirements catalogue [a document] that lists requirement instances along with properties such as reference number, description, source, owner, priority, and requirement type. (E.g. throughput = 100 tps, response time = 1 second.)

Requirement type [a concern] a kind or group of needs that is generally applicable to system structure or behaviour. It usually corresponds to a concern that is held by stakeholders or is addressed in viewpoints. (E.g. throughput, response time.)

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

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

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. Audit requirements have implications for data records and retention.
>
> **Regulatory requirement** [a requirement type] including legislation and regulations that direct or constrain architecture work.
> - IT accountability and procurement: Regulation that makes public sector, IT directors and CIOs accountable for justifying investment in IT and for fair procurement from suppliers. E.g. Information Technology Management Reform act (ITMRA) of 1996, Division E, P.L. 104-106. This "Clinger Cohen Act" was the stimulus for many early enterprise architecture initiatives.
> - Data protection: Legislation that directs an enterprise to protect data from unauthorised access. E.g. UK's data protection act.
> - Data freedom: Legislation that directs an enterprise to make data available to people who request it. E.g. Freedom of Information Act 2000.
> - Disability and accessibility: Legislation that directs enterprise to build systems that can be accessed by any member of the public. E.g. UK Equality act. US Americans with Disability act. W3C Web Content Accessibility Guidelines.
> - Shareholder protection and audit: Legislation that directs an enterprise to maintain records showing how it has directed, monitored and controlled its business. E.g. US Sarbanes-Oxley act of 2002. Basel II.
> - Intellectual property rights: International and national laws protect people and enterprises from theft of intellectual property.
>
> **Standard** [a concern] a widely-accepted definition of a structure, process or rules, intended to increase uniformity and interoperability between distinct systems and processes.
>
> **Enterprise Standards Information Base** [an artefact] a catalogue of standards that is recommended or used across the enterprise. (Cf. The Open Group's SIB.)

2.5: Scoping

Solution architecture description [an architecture description] describing a system or other solution to problems and requirements. Its elements may be mapped to directives, aims and plans. It is usually defined at several levels of abstraction.

> **Solution concept/vision** [an architecture description] that briefly describes of a target, just enough to enable options to be compared (including any initial idea of costs and risks) and solution outline work to proceed. It may be a response to a business problem or an elaboration of how to reach a business vision.
>
> **Solution outline** [an architecture description] that describes the high-level design of a target system, produced after a first pass architecture description, enough to enable an acceptably accurate estimate of cost, benefits and risks, and projects to be planned.
>
> **Solution to be built** [an architecture description] that describes a project-ready architecture, or detailed design, that is complete enough for the project to be scheduled and resourced, and the building team to start work.

Scope view [a view] a dimension of scope.

- Breadth: scope of the enterprise, system or solution (see below).
- Domain in focus: business, application or infrastructure change (see section 1).
- Depth: the detail to which description or design should be completed
- Constraints on work (see below).

Breadth of enterprise or system [a scope view] that may be defined in several ways.

- Aim view: goal/objective/requirement catalogue
- Service view: a service catalogue.
- System view: a top-level context diagram(see below).
- Process view: a top-level process map or use case diagram.
- Data view: a conceptual/domain/business data model.

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

Constraint on work [a scope view] a factor such as time, budget and resources that limits work to be done, or potential options.

2.6: Business context

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, stages or projects within the overall solution.

See the Migration Planning section for more on business cases.

3: Architecture frameworks

A comprehensive architecture framework contains advice on processes for architecting, products for architecture description, and people (architect roles).

However some frameworks are limited to architectural description - to conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders.

3.1: Foundation (rarely examined)

Architecture state [an architecture description] 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.

TOGAF: **The Open Group Architecture Framework** [an architecture framework] for transforming an enterprise architecture from a baseline state to a target state. It is published on a free-to-read public web site, though its use by a commercial organisation is restricted by copyright conditions. It is centred on a process called the architecture development method (ADM).

ADM: **Architecture Development Method** [a process] defined in TOGAF to develop and use an enterprise architecture. It involves 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
- H: Architecture Change Management.

AM: **Avancier Methods** [an architecture framework] focused on solution architecture, though it also addresses enterprise architecture rationalisation. It is published on a free-to-read public web site, though its use by an organisation is limited by copyright conditions. It features a solution architecture process with four phases (Initiate, Architect, Plan and Govern), each subdivided into lower level processes.

3.2: Architecture description

Architecture content framework: [a passive structure] for organising an architecture description composed of deliverables, artefacts and entities.

> **Architecture deliverable**: [a document] that architects produce or contribute to, for approval by sponsors if not all stakeholders. It should conform to a document type defined by a standard contents list. Deliverables often contain artefacts, which are in turn composed from architectural entities.
>
> **Architecture artefact**: [a model] that conforms to one of three artefact types: catalogue, matrix and diagram.
>
> **Architecture entity**: [an entity] in an architecture description meta model. One entity instance can appear in several artefacts, which can appear in several deliverables.

Mapping [a correspondence] that is drawn between elements of the same or different structures. Correspondences can be mapped for several purposes including:

- gap analysis (see section 10),
- impact analysis (see section 11),
- requirements traceability analysis
- cluster analysis (see section 4).

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

Viewpoint [a work product description] that typifies a view and provide a template for it. 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 description, 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 description] a description that simplifies or abstracts from a thing or another description. It displays or records some properties of what is modelled and enables some questions to about it to be answered.

3.3 Abstraction of architecture descriptions from systems

Enterprise architecture is abstracted from solution architecture, which is abstracted from software architecture and detailed design, which are abstracted from building work.

Abstraction and its opposites.

> **Abstraction** [a technique] by which a simpler description is derived from other descriptions or from reality.
>
> **Refinement** [a technique] that yields a detailed description that conforms to a more abstract description. Everything in the abstraction holds, perhaps in a somewhat different form, in the refinement.
> **Concretion** [a technique] that instantiates a description as one or more real, active (or activatable) components.

Omission v elaboration.

> **Omission** [a technique] that removes details or perspectives from a description.
>
> **Elaboration** [a technique] that adds details or perspectives to a description.

Composition v. decomposition

> **Composition** [a technique] that assembles parts into a whole and/or hides components behind a façade. Architects describe systems in terms of coarse-grained components, processes and services.
>
> **Decomposition** [a technique] that divides a whole into parts and/or identifies components behind a facade. The conventional advice is that it is difficult to maintain the integrity of a hierarchical structure that is decomposed more the three or four levels (or more than a thousand elements) from the top.

Generalisation v. specialisation

> **Generalisation** [a technique] that defines properties shared by subtypes or entities. Architects look to maximise re-use of common components, processes and services across the enterprise.
>
> **Specialisation** [a technique] that extends or modifies generic properties to define a subtype or smaller population of entities. It can also mean configuring an instance by selecting variable values from a general range.

Idealisation v realisation

> **Idealisation** [a technique**]** that reverse-engineers a more logical description by omitting some details relevant to a particular physical form of the thing described. Architects produce and work with logical descriptions of components, processes and services.
>
> **Realisation** [a technique] that forward-engineers a more physical description. It adds details relevant to a particular physical form of the thing described. OR concretion: the instantiation of a description as one or more active, run-time, components.

3.4: Architecture models and languages

Idealisation hierarchy: the classic hierarchy of conceptual, logical and physical model.

> **Conceptual (or domain) model** [an artefact] an abstract logical model that defines terms and concepts in a business or problem domain without reference to any computer application.
>
> **Logical model** [an artefact] a model of a particular system that excludes details of that system's physical implementation.
>
> **Physical model** [an artefact] a model of a particular system that is vendor or technology specific and/or includes details of its physical implementation.

MDE: **Model-Driven Engineering** [a technique] used in methods and tools for transforming a conceptual model to a logical model, and a logical model to a physical model, and the reverse. It covers forward engineering and reverse engineering. For example, Model-Driven Architecture.

Model-Driven Architecture (MDA) [a model-driven engineering technique] a vision of the Object Management Group (OMG) that encourages vendors to develop tools for Model-Driven Engineering to standards defined by the OMG. The idealisation hierarchy is:

- computation-independent model (CIM),
- platform-independent model (PIM), unrelated to a specific technology
- platform-specific model (PSM), related to specific infrastructure technology.

Modelling language [a standard] that defines shapes for representing architecture entities and arc/line styles for representing relationships between them. Three international varieties are IDEF, UML and ArchiMate.

> **IDEF**: **Integration DEFinition language** [a modelling language] in the field of systems and software engineering, originally funded by the US DoD. Its most-well-known notations are IDEF0 (a process modeling language building on SADT) and IDEF1X for information models and database design.
>
> **UML**: **Unified Modelling Language** [a modelling language] maintained by the Object Management Group. It was designed to help in OO software design, though often used outside of that. It includes structural models such as class diagrams and deployment diagrams, and behavioural models such as use case, activity and sequence diagrams.
>
> **ArchiMate** [a modelling language] maintained by the Open Group. It was designed to help in architecture description. Components, interfaces and services are shown in distinct boxes. It overlaps with UML but is intended for more abstract design.

3.5: Pre-defined classifications and reference models

Architecture repository [a data store] an information base used by architects; a system that holds and manages all the meta data that describes an enterprise and its information systems. Its structure is defined in some kind of schema or architecture meta model. The content of the repository can be categorised using, for example, the Zachman Framework or Enterprise Continuum.

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. 3.5a The Zachman Framework

The 6 columns, though titled with interrogative questions, are mapped to architectural description elements. The 6 rows are primarily levels of idealisation-realisation from context to operational systems, but are mapped to stakeholder types and architecture domains or views. Zachman says the rows should not be interpreted as levels of decomposition.

Enterprise continuum [a pattern] a logical structure for classifying and organising architecture description artefacts. It is a core part of TOGAF. 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. 3.5b 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 domain/view/layer

4.1: Foundation (rarely examined)

Required behaviours	Logical structures	Physical structures
Business service Business process	Business function Role	Organisation unit Actor

Fig. 4.1a Base business architecture concepts

The base elements in this domain are explained in later sections.
This first section introduces some background concepts.

Enterprise [a system] a business or organisation, in the public or private sector, with common goals and budget. It is directed and controlled by a management board. Its players cooperate to meet agreed goals and objectives. It requires human and other resources (materials, energy and information). It is usually the highest level of an organisation. It usually spans several organisation units.

Business domain classifies an enterprise or organisation unit by the services it offers or the expertise it has. The primary business function(s) of an enterprise or a component thereof. (E.g. law, employment law, telesales, insurance, airline operation, airline maintenance, security, emergency response.)

Mission [a driver] what an enterprise, business or organisation is about; its reasons for being; the essential products and services it offers customers.

Vision [an aim] what an organisation wants to be or become;"a high-level outline of an aspirational target state for an enterprise (a business rather than solution vision). The state may be associated with measurable aims, or only a general direction for planners to follow.

Business model [a description] that commonly means a top-level summary document showing how an enterprise delivers money or other value to its owners or sponsors. But in EA, it usually means part or all of a business architecture description.

Operating model [a description] that commonly means a business model. But in EA, it usually means a "strategic co-ordination view", as defined in "EA as Strategy" by Ross, Weill and Robertson

Strategic coordination view [a description] that shows the degree to which an organisation aims to standardise and/or integrate business processes and business data. It can be positioned in a two-by-two grid of standardisation against integration.

High Integration	**Coordinated**	**Unified**
Low Integration	**Diversfied**	**Replicated**
	LowStandardisation	**High Standardisation**

Fig. 4.1b Strategic coordination view

4.2: Business structure

Actor [a physical business component] or individual able to play one or more roles in the performance of processes. Some limit the term to human actors (a sales executive, a customer). Some use the term also for organisation units and/or applications.

Role [a logical business component] defined as one or more services or processes that can be realised or performed by one actor (e.g. loan applicant, expense claim approver), perhaps along with abilities required to play that role.

Organisation structure chart [an artefact] showing a structure of organisation units and/or reporting lines between managers; usually a hierarchy. It may also show human actors or activities assigned to each organisation unit.

Organisation unit [a physical business component] or individual node in a management structure, able to fulfil one or more functions. It should have goals and objectives with measures, a manager, a budget, and other resources to fulfil its functions.

Functional decomposition (aka capability map) [an artefact] that shows an idealised or logical organisation structure. Enterprise architects strive to define a stable structure that can be used as a basis for business analysis, heat mapping and classification of other architectural entities.

Business function [a logical business component] defined as one or more services or processes that are cohesive in some way, perhaps along with abilities required. It is a logical subdivision of an enterprise's overall capability. It is realised by one or more organisation units and actors.

> **Capability:** a variously-defined concept that usually corresponds to a business function, but may embrace organisation units and actors who fulfil the function, or mean other things.

EBF: Elementary Business Function [a business function] a node at the bottom of a function decomposition, typically at the 3rd or 4th level of decomposition. If the structure is decomposed far enough, each EBF should correspond to an EBP.

Core business function [a business function] that is focused on the development, marketing, sales, creation and delivery of business products and services.

Support business function [a business function] that serves core business functions. Being similar in different businesses, it is a candidate to be out-sourced and/or delegated to a "shared service". E.g. personnel, procurement, finance or facilities management.

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

Business network diagram [an artefact] that shows where business components (units, functions, roles or actors) interoperate by requesting and providing flows or services. Each service delivers a result, and output or product of value to the service requester or receiver. The components and services can be modelled at any level of abstraction you choose.

Business interface [an interface] a collection of business services provided by a business component (function or role). It identifies services, may provide access to them, and hides processes and resources need to deliver the service.

SLA: Service Level Agreement [a document] that records a business interface between a service provider and its customer(s). A contractual document that defines the legal context for service delivery. It may include a schedule of discrete services to be delivered, with agreed-to service levels.

Cluster analysis [a technique] for gathering related items into groups. It is used to group elementary business processes (EBPs) into a business function. It is also used in exploratory data mining.

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

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

Business data model [an artefact]. See the Data domain section.

4.3: Business behaviour

Value chain diagram [an artefact] that shows the top-level core and support functions of a business, and may suggest an end-to-end behavioral flow as well (usually in the arrow shape first drawn by Michael Porter in "Competitive Advantage" 1985.)

Process map [an artefact] that shows a top-level view of named business processes. Processes may not be connected or else related very informally. It may show coarse-grained functions (structural nodes) in vertical columns and show processes (behavioral flows) left-to-right.

Business behaviour [a service or process] triggered by an event that follows rules definable in the form of preconditions and post conditions. A repeatable activity that (if successful) ends with a result/output/product of value to the service requester or recipient.

> **Business service** [a service] that can be requested of a business, or a component thereof, by its customers, suppliers or employees (e.g. order, payment, report, weighing.) An external view, a logical contract, that encapsulates whatever process(es) are need to deliver the desired outcome.
>
> **Business process** [a process] that is performed by the actors in a business in the course of delivering a business service.

Value stream [a process] "an end-to-end collection of activities that create a result for a 'customer' who may be the ultimate customer or an internal 'end user' of the value stream" Martin (1995). It may be drawn as a rich picture rather than a process flow chart.

Value stream analysis [a technique] that involves measuring process steps, and optimising the process in the light of those measurements. It can help people remove wasteful activities and optimise processes in product manufacturing industries, where each activity should add more value to the end product or service than it costs.

Process decomposition hierarchy [an artefact] in which each step or activity in a higher-level process is elaborated and defined as a lower-level process of several activities.

EBP: Elementary Business Process [a process] an activity at the bottom of a business process decomposition hierarchy.

OPOPOT [a technique] one person, one place, one time; a rule of thumb used to define the bottom level of business process decomposition.

Structured analysis verification [a technique] to ensure a business architecture description is comprehensive and consistent. In theory, every EBP can be matched to an EBF. In practice, a structural decomposition view usually stops short of the detailed activities documented in process flows, but some cross-validation can still be useful.

Process automation [a technique] where by a business processes is decomposed to level where application use cases can be identified, which are further decomposed to a level where automated services can be identified.

Business scenario [an artefact] that outlines a process, along with the human and computer actors involved in the process steps. It is useful in creating and presenting an architecture description. It may be defined to support a solution vision or business case. It may be defined during business architecture description. It may be presented as an example instance of a business process.

Business scenario-driven design [a technique] that proceeds by defining the main elements of a process, scenario or value stream:

- The aims (outcomes or effects) and outputs (products or services)
- The main process steps
- The human actors (roles) involved in process steps.
- The computer actors (roles) involved in process steps.

5: Data domain/view/layer

5.1: Foundation (rarely examined)

	Data in motion	Data at rest
Data object	Data event	Data entity
Data container	Data flow	Data store

Fig. 5.1 Base data architecture concepts

The base elements in this domain are explained in later sections. This first section introduces some background concepts.

Structured data [a data object] a data store or flow that fits a pre-defined data structure. It is composed of data items. It usually records real word entities or events. It may contain references to unstructured data. All popular architecture framework focus on structured data.

Unstructured data [a data object] text or images that do not fit a pre-defined data structure, as in emails, voice and video. It may contain recognisable structured items.

Data item [a data object] an elementary unit of information, a fact. An attribute of a data entity or data event. A variable containing a value that is constrained by a data type.

Meta data [a description] of data. It may take the form of a data structure, data type, constraint rule, derivation rule or other data quality.

> **Data structure** [a type] a structure that arranges data items in one or more groups. It may be described as a data model or a regular expression.
>
> **Data type** [a type] that defines the properties shared by instances of a data item or data entity. 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.
>
> > **Primitive data type** [a data type] pre-defined in a programming language. e.g. character string, integer, Boolean, floating-point number (decimal).
> >
> > **User-defined data type** [a data type] defined by analyst or architects that is bespoke to the business at hand. It may be simple (e.g. credit limit, order value) or complex (e.g. address, tax reference number, order, product).
>
> **Data rule** [a business rule] recorded in a rule repository, data dictionary or data model as a constraint or derivation rule.
>
> > **Constraint rule** [a data rule] that limits the values of a data type.
> > **Derivation rule** [a data rule] that defines how a data value is derived from one or more other values (a special kind of constraint).

Data dictionary [an artefact] 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

Data at rest [a viewpoint] relating to data that persists.

Data entity [a data object] composed of data items that represent facts about a discrete business entity or event. It may be specified at a conceptual, logical or physical level. It may be mapped to data stores and/or data flows input to or output from IS services.

Data store [a platform component] that holds a persistent data structure. A cache, file or database from which data can be extracted by an application. The content of a persistent data store can be defined in a data model.

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

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

Data model [an artefact] that shows the content of a data store. A structure of inter-related data entities.

> **Business data model** [an artefact] that names and describes things in the business world that business people need to remember, regardless of computing. It identifies data entities that may appear in several data stores. It may define some business terms and concepts, but usually excludes details such as data types.
>
> **Logical data model** [an artefact] that shows the data structure that must persist for the processes of an application to work. It shows relationships between instances of data types. It is usually drawn as a normalised data structure.

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

Normalisation [a technique] for defining a data store structure that assists data integrity by storing each fact once. It also optimises update processes by minimising redundant data storage. The outcome of relational data analysis.

De-normalisation [a technique] that optimises input and/or output processes by structuring a data store structure to reflect the most important input or output data flow structures, at the expense of duplicating some stored data.

Data warehouse [a data store] that is optimised for the production of management information reports. It usually holds a non-normalised data structure.

Data server [a platform component] that hosts a database management system and enables a data store to be accessed by applications. It usually enables direct access to any data entity instance in the data store, using its primary key.

5.3: Data in motion

Data in motion [a viewpoint] relating to data flows.

Data event [a data object] representing an event in an input or output data flow.

Data flow catalogue [an artefact] that lists the data structures transported from senders to receivers within a given application portfolio or system family. A data flow can take the form of a message, file, report, form, display format or other data stream.

Regular expression [a model] that describes the structure of a data flow (as a logical data model describes a structure of a data store. It is drawn using the universal grammar for defining the structure of a data flow or message. It is a hierarchical structure of in which every element is part of a sequence, or an option of a selection or an occurrence of an iteration.

Data format [a standard] for the organisation of a data structure, such as Comma Separated Values (CSV), JSON or Extensible Mark Up Language (XML).

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

Canonical data model [a standard] that provides the "one true definition" of data types used by an enterprise. It is an important tool in SOA and application integration. 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.

5.4: Data qualities and integration

Data quality [a property] of a data item, data structure or data store. Notably, 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. E.g. 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.

Data dissemination view [an artefact] that maps data entities to the applications, data stores or locations that hold them. This view shows duplication of data between data stores. It is useful in analysis of change impacts, data mastering and security vulnerabilities.

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 domain

How applications are modularised and interoperate is important to enterprise architecture.

6.1: Foundation (rarely examined)

	Required behaviours	Logical structures	Physical structures
Software	Operation	API	Application component

Fig. 6.1 Base software architecture concepts

Operation [a service] that can be requested of a software component. An abstract or virtual operation is an operation for which the signature is defined, but not the internal procedure that implements the operation.

API [an interface] a collection of automated services accessible by software clients; it identifies services, may provide access to them, and hides what performs.

Application component [a component] module or object capable of offering automated services. Module is a generic term. Class is the OOP term. An object is an instance of a class; it has identity, behaviour and state; it can perform the general operations of the class and maintain object-specific data item values in its state.

Component role: a role such as client or server, sender or receiver, prior or successor, played by one component with respect to another.

> **Client component** [a component] that requests one or more services from a server component. The request is usually called an invocation.
>
> **Server component** [a component] that can provide one or more services in response to requests from a client. Cluster analysis can be used to group services that are related (e.g.) by access to the same data) into one API or one server component.

Delegation [a process] whereby on component calls on another to do the work, invoking it by passing a message.

Dependency [a mapping] a relationship between two parties in which any change to the depended-on party implies an impact analysis of the dependent party. Often reflects a client-server invocation, or delegation to a server. Often represents a run-time relationship.

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

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

Class diagram [an artefact] that shows the design-time structure of a software application, showing which components are related and how.

Sequence diagram [an artefact] that shows how components cooperate at run-time to enable a process. Knowing how a design-time structure will behave at run time is critical to meeting requirements.

6.2: Modular design and integration

This section reflects a history in which there has been a drift from **closely-coupled** to **loosely-coupled** software architecture.

Modular design [a technique] for dividng a "monolithic" system into self-contained, encapsulated components. Concerns include: scoping a component, avoiding duplication between components, separating or distributing components, and integrating components.

Local Procedure Call [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 [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.

OO Programming Language: a language in which components differ from more traditional modules in three ways.

- It is possible to deploy many instances (objects) of a module type (class).
- A client uses an object identifier to locate the module instance (object) it wants.
- Objects of one class can inherit/extend the operations of a more generic class.

OO Design [a technique] a kind of modular design and integration that *initially* tended to assume the following features.

1. a component is an object, an instance of a class, identified by an object identifier
2. client/sender and server/receiver objects work in the same name space
3. an object is stateful
4. an object can reuse behaviour by inheriting it from a super type class
5. intelligent domain objects (rather than intelligent process/control objects)
6. client objects communicate with server objects by request-reply invocations
7. a server object can accept only one invocation at once (so blocks others)

Garbage collection [a process] that deletes unused stateful objects from memory. (Multi-user business systems, handling millions of entities and events, are unlike the earliest OO programs, which needed to maintain only few stateful objects in memory.)

DO: **Distributed Objects** [a technique] a distributed modular design and integration style that employs an Object Request Broker (a kind of middleware).

Object request broker (ORB): Like RPC with extra features. It enables the objects of an OO program to be distributed. Software is coded as though all objects are on one computer. The ORB handles the distribution of objects between computers. So (in theory) the distributed system behaves like one OO program. It may add transaction management, security and other features. OMG's CORBA emerged as the standard.

CBD: **Component-Based Design** [a technique] a modular design and integration style that involves dividing a system into encapsulated components that are much more coarse-grained than the objects in conventional OO design, perhaps in different places and technologies.

IDL: Interface Definition Language: a language for defining an API that is supposed to be independent of the technology used to implement the component behind the interface.
It enables communication between components in different languages and running on different operating systems.

WSDL: Web Service Description Language: [a standard] from the W3C for defining an interface, including a signature, protocol and web address for each discretely accessible operation/service. Initially, related to XML and SOAP, now usable with JSON and HTTP.

SOAP: A W3C standard protocol, initially devised by Microsoft as a means to connect remote components without using an ORB. It allows components deployed on disparate operating systems (e.g.Windows and Linux) to communicate by sending XML messages over HTTP.

Web Service: [a component] that can be invoked over "the web" using an internet protocol and a published interface. It uses open standards like WSDL, XML and SOAP, but no particular standard is widely agreed as constraining what the term means.

SOA: **Service–Oriented Architecture** [a technique] a modular design and integration style that is a more loosely-coupled than Distributed Objects and facilitates the reuse of remotely accessible services. It is often associated with the use of Web Services, but does not have to be.

REST: **Representational State Transfer** [a technique] a modular design and integration style 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 supports clients wanting to invoke operations on entities in a remote data store. Client apps that speak OData can easily connect to data server apps that provide CRUD operations on a logical data model.

EDA: **Event-Driven Architecture** [a technique] a modular design and integration style decouples the senders of news or update events from the receivers. Any component can receive or read any event/message published or posted by any other component. It uses either the publish and subscribe functions of a middleware technology, or else a shared data space.

6.3: Decoupling

Loose coupling factor [a property] considered in deciding whether components should be tightly or loosely coupled.

Synchronous [a property] 1: A request-reply style; a client must wait for a server to reply before continuing. 2: A blocking style; a server serves one client at a time, turning away any other client who attempts to request a service. The caller and responder hold a channel open, blocking others from using it.
Aside: 1: is the usual invocation from one COBOL module to another or one Java object to another. 2: is the usual invocation style in CORBA.

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

The table below lists decoupling techniques.

Factor	Tight coupling	Decoupling techniques
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
Time	Synchronous request-reply Blocking servers	Asynchronous messaging Non-blocking servers
Location	Remember remote addresses	Use brokers/directories/facades
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 structures and patterns

Façade [a pattern] a frontage to a building. An interface to system behaviour that shields clients from some kinds of change to the system. 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 processing of persistent data (model).

Proxy [a pattern] a surrogate for a distributed component. It is used in distribution of code between different name spaces, as in “Distributed Objects”

Observer [a pattern] a component that monitors the state of another component. A primitive kind of “Event-Driven Architecture”

Singleton [a pattern] a component (or class) with only one instance (or object).

7: Applications domain/view/layer

7.1: Foundation

Required behaviours	Logical structures	Physical structures
IS/App service	Application interface	Application

Fig. 7.1 Base applications architecture concepts

Application [a component] of business-oriented software (e.g. CRM, Billing). It is specified logically by the IS services it provides, and sometimes also by the data entities it maintains, and/or physically as a vendor/technology specific product that can be hired, bought or built. It is divisible into three categories: platform, generic and business application.

Application interface [an interface] a collection of application services accessible by application clients; it identifies services, may provide access to them, and hides what performs them. It may be defined in a user interface or API.

IS/App service [a service] that can be requested of a business-oriented application component, by a human actor or another application component. It could be a use case provided by one application to an end user, or a fully automated service provided by one application to another application.

IS/App service examples include: check customer credit, provide weather data, consolidate drilling reports, create policy, pay premium, register claim. Note that these digital services might be wrapped up inside wider business services of the same name.

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. Two application classifications follow.

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. E.g. accounting; billing, customer relationship management, enterprise resource planning, business intelligence, 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. E.g. calculator, drawing tool, groupware, media player, spreadsheet, browser, word processor.

- **Platform application**: [an application] or system software that runs computer hardware or serves other applications. See section 9 for more detail.

Classification by business function [a pattern] dividing applications by business function.

- **Enterprise Resource Planning (ERP)** [a business application] that supports the planning of how enterprise resources (materials, employees, customers, etc.) are acquired, moved from one state to another. It maintains data needed for some or all of Manufacturing, Supply Chain Management, Financials, Projects, Human Resources, Data Warehouse and Management Information. It can include CRM and Billing.

- **Customer relationship management (CRM)** [a business application] that supports the development and maintenance of mutually beneficial long-term relationships with customers. It helps with some or all of the following attracting customers, transacting business with customers, servicing and supporting customer, enhancing customer relationships.

- **Billing** [a business application] that sends bills and matches payments received to bills sent.

- **Business intelligence** [a business application] that extracts management information from large quantities of data.

7.3: Application structure

Application portfolio catalogue [an artefact] listing business applications and recording their properties. Usually structured so as to reflect the business function hierarchy.

Applications communication diagram [an artefact] 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.

Application / data entity matrix [an artefact] that maps data entities to the applications that create and use data. It may be completed with Create, Read, Update, and Delete entries.

7.4: Application behaviour

Process realisation diagram [an artefact] 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.

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

IS/App service [a service] that can be requested of a business application; a use case provided by one application to an end user, or a fully automated service provided by one application to another application. (REPEAT)

> **Use case** [an IS/App service] at the human-computer interface. A use case description defines a use of a system by an actor, typically in the course of an OPOPOT business process or role. It is normally named as a goal in verb-noun form (e.g. assess claim). It is defined primarily by user role, trigger event, inputs and outputs, preconditions and post conditions, and non-functional requirements - and secondarily by its process flow in the form of the main path and alternative or exception paths. The details of each process step (including automated services or transactions invoked) may be documented separately from the use case description.
>
> **Automated service** [an IS/App service] that can be requested of a software component. It is sometimes an ACID transaction.
>
> > **Automated business service** [an automated service] a kind of automated service whose input and output data is defined in a canonical data model.
> > Aside: It is the interface that matters, not the deployment location. So it can be provided by a broker application or an application that encapsulates a data source.
> > **Automated data service** [an automated service] a kind of automated service whose input and output data items are defined according to the parochial or physical data model of a specific data source.

Transaction [a process] 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] a unit of work that is Atomic, Consistent, Isolated and Durable. It can be rolled back if a specified precondition is violated. Using a transaction manager to automate transaction roll back preserves the integrity of stored data and saves considerable design and development effort, but is impossible in loosely-coupled designs. Where a process has an update or output effect that cannot be rolled back, then compensating transactions may have to be designed.
>
> **Compensating transaction** [a transaction] a process 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, report cases of data disintegrity.

7.5: Application Communication Patterns

Application communication style [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.

> **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.
> Aside: The technologies do what email infrastructure does for people, that is, enable them to communicate asynchronously via messages - rather than talk directly over an end-to-end network connection kept open for that conversation.
>
> **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.
>
> Aka “shared memory”, “space-based architecture” or “blackboard design pattern” or “passive mediator”.

Aside: Different communication styles may be used at different levels of a communication stack. Under the covers, all communication requires point-to-point connection at some level.

7.6: Applications Integration Tools (rarely examined)

ETL tool: (a platform application 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: See “RPC” and “ORB” in Software Architecture

Web Service: See “Web Services” in Software Architecture.

ESB middleware: (a platform application component) that may:

- store, route and forward messages between distributed components - using message queues
- 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.

7.7: Applications Integration Patterns (rarely examined)

Application integration pattern [a pattern] for sharing data between applications or their databases, including those listed below.

Point-to-point integration [a pattern] in which subsystems talk to each other directly.

Hub and spoke integration [a pattern] in which subsystems communicate via a mediator.

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

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

Physical master data [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 [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 apps: present user interfaces, capture events from them and invoke broker apps.
- Broker apps: decouple by providing automated business services to user apps, and invoking data services from data app(s)
- Data apps: provide automated data 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 (NFRs)

This section contains a selection of measures and common techniques that could be relevant in answers to exam questions.

8.1: Non-functional requirement types (NFRs)

Performance [a requirement type] that is subdivided 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. Sometimes, improving one measure damages the other.

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. Possible measures include MTBF / (MTBF + MTBR), which usually refers 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).
Aside: The measures above are sometimes applied only to platform applications, ignoring faults and failures in business applications.

Recoverability [a requirement type] the ability of a system to be restored to live operations after a failure. Possible measures include mean time to repair (MTBR). Usually refers to disaster recovery using resources at a remote site.

Integrity [a requirement type] a term with four possible meanings defined under "Data Integrity".

Scalability [a requirement type] the ability for a system to grow with increased workloads.

Security [a requirement type] the ability to prevent unauthorised access to a system.

Serviceability [a requirement type] the ability to monitor and manage a system in operation.

Usability [a requirement type] the ability of actors to use a system with minimal effort.
Aside: Measures include PLUME. Productivity; tasks completed in a given time. Learnability; how much training to reach a proficiency level. User Satisfaction; scores given by users. Memorability; how long to forget how to use the system. 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 for systems to exchange data using shared protocols and networks. It may embrace also "integratability" - the ability of interoperable systems to understand each other, which requires either common data types or translation between data types.

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

8.2: Design for NFRs (bar security)

Design for response time [a technique] that may involve minimising distribution, network hops, database accesses, and message queue length. The general advice is to look for bottlenecks and tackle them one by one.

Design for throughput [a technique] that usually involves running processes in parallel.

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

Scale out (aka clustering) [a technique] that increases the number of parallel nodes, sometimes adding more nodes to a cluster. Usually requires some kind of load balancer to sit in front of the cluster and distribute service requests between them. Generally good for throughput. Not always good for response time.

Caching [a technique] that places copies of persistent data in a location nearer to the user than the original data source. Generally good for response or cycle time. Can raise concerns about data integrity and security.

Database optimisation [a technique] for reducing needless database access that include normalisation, denormalisation, indexes and access path analysis.

Indexing [a technique] that creates a list of pointers to stored data elements. Indexes can help to optimise batch input and output processes, which typically run overnight. Indexes may be temporarily disabled during the day to optimise on-line update processes.

Access path analysis [a technique] that studies the route a process takes through a data store structure. A very common source of performance problems is that an SQL programmer does not know the access path their procedure takes through a database. So it is advisable to use access path analysis and/or employ highly skilled SQL resources for critical database access programs.

Design for resilience (availability and reliability) [a technique] that builds redundancy into the system, with parallel active components or fail over to a passive standby component. E.g. to scale out, or add one to the number of servers in cluster that calculation or prototyping suggests is needed. Other techniques include defensive design

Fail over [a process] 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.

Defensive design [a technique] that means either 1. Designing a client component so that it does not fall over if a server component does not work properly; asynchronous invocation may help. Or 2. Designing a server component so it does not depend on input data being valid, which means testing input data and preconditions before processing (and is the opposite of “Design by Contract” as promoted by Bertrand Meyer.)

Design for recoverability [a technique] to back up and provide some kind of switch-over or fail-over procedure. Procedures must address also “fail back”, to return operations from a disaster recovery site to the normal production site.

Backing up [a process] to make a copy of data, so that it can be used to restore the original after data loss. Used in disaster recovery. Also used to restore individual files that have been

deleted or corrupted. Backups are typically the last line of defence, coarse-grained and can be inconvenient to use.

Backup site [a location] where systems are or can be duplicated. A cold site has no equipment. A warm site has infrastructure but no up-to-date data or software. A hot site has up-to-date software and more or less up to date copies of data.

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 instrument applications so that they report on what they are doing, and how well they are doing it.

8.3: Design for Security

Security architecture [a view] showing design features designed to protect a system from unauthorised access - to maintain the required data qualities of confidentiality, availability, and integrity. Not a cohesive architecture domain on its own so much as features added to other views.

Design for human and organisational 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.

Data security [a concern] 1: Confidentiality alone. 2: A combination of Confidentiality, Integrity and Availability. 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.
>
> **Security policy** [a document] that defines which actors have access rights to which data objects in a given information domain – and which security processes or properties are used.
>
> **Identity** [a property] one or more data items (or attributes) that uniquely label an actor. E.g. 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 applications security [a technique] for preventing unauthorised use of an application.

> **Identification** [a process] via which an actor supplies their identity to an authority. It is usually followed by authentication.
>
> **Authentication** [a process] to confirm or deny that an actor is trusted - is the entity to which an identity was given. E.g. A password check. 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 infrastructure security [a technique] for protecting client and server computers from malicious access.

> **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] normal HTTP interoperation over an encrypted Secure Sockets Layer (SSL) or Transport Layer Security (TLS) connection. This ensures reasonable protection of data content from those who intercept the data flow 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 with the web server cannot be read by eavesdroppers.

9: Infrastructure domain/view/layer

9.1: Foundation (rarely examined)

Required behaviours	Logical structures	Physical structures
Platform service	Platform interface	Platform application, Node, Network

Fig. 9.1 Base infrastructure architecture concepts

Platform service [a service] that can be requested of a technology component by an application component or another technology component. E.g. transaction roll back, or user authentication.

Platform interface [an interface] a collection of platform 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.

Platform application [a component] of generic infrastructure software. It is specified logically by the platform services it provides, and/or physically as a vendor/technology specific product that can be hired or bought.

Examples: operating system, device driver, web server, data server, windowing system and message broker; also programming language and compiler. It may serve general non-functional requirements: e.g. identity management, data replication etc.

Node: [a component] a computational resource upon which artifacts may be deployed for execution. Nodes can be interconnected through communication paths to define a network structure or topology. Nodes can be virtual or physical servers. Nodes can be nested.

Communication network [a structural view] of communication paths that enables computers and/or electro-mechanical devices to send and receive data.

9.2 Enterprise technology rationalisation

Application / technology matrix [an artefact] that maps business applications to the platform applications and/or nodes they depend on.

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

ETC: Enterprise Technology Classification [a passive structure] for the organisation of a technology portfolio catalogue.
E.g. Client (user access) devices. Generic user applications. Operating systems. Database management. Middleware. Software development. Servers. Data storage. Networks. IT services management / operations. Environment. Security.

> **OS: Operating System** [a platform application] designed to enable programs to run on a computing device. (E.g. Windows, Windows Azure, Linux variants (SUSE & Red Hat), Unix variants (IBM AIX, HP UX, Oracle Solaris), IBM i (ex i5/OS, ex OS/400 on AS/400)).
>
> **DBMS: Database Management System** [a platform application] designed to enable programs to store and retrieve persistent data. (E.g. DB2, SQL Server, Oracle, Sybase and Teradata.)
>
> **Middleware** [a platform application] designed to assist the interoperation of distributed applications components. (E.g. Message broker, Enterprise Service Bus, Transaction processing manager.)

TRM: **Technical reference model** [a passive structure] for organising the platform services provided by infrastructure technologies to applications. It may share its top-level structure with the ETC (above). It can provide a requirement specification for technology rationalisation.

Technology rationalisation [a technique] for studying the services provided by a baseline technology infrastructure and defining a de-duplicated target architecture.

- Classify baseline platform applications (see ETC)
- Catalogue baseline technologies (see Technology portfolio catalogue).
- Classify baseline platform services (see TRM)
- Catalogue baseline platform services
- Define target platform services
- Define target platform components (platform applications and nodes)
- Plan baseline-to-target migration
- Govern delivery of the change.

Virtual machine [a platform application] of software that enables application programs to run above – decoupled from - the underlying operating system and/or hardware processor. It enables applications to be moved between different operating systems and/or processors. It enables server consolidation.

Server consolidation [a work process] a programme of work to deploy currently deployed applications to fewer servers, usually involving virtualisation.

9.3: Solution technology elements and definition

Device [a node] with processing capability upon which artifacts may be deployed for execution. E.g. 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 artifacts. E.g. 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 artifact: a development deliverable (source file, script, executable, database table etc.) that is deployable to a node instance. It may be nested, so a component and its descriptor can deployed inside one artifact instance to a node instance.
Deployed artifact: an artifact that has been deployed to one or more deployment targets
Deployment: the allocation of one artifact to one deployment target (defined at the type level or instance level).

Software deployment diagram [an artefact] that shows deployments of application and other software components to execution environments and nodes (virtual and/or physical).
Hardware configuration diagram [an artefact] that shows the devices and indicates where they are connected by communication paths.
Communications engineering diagram [an artefact] that shows devices, and is focused on detailing the network components, sometimes annotated with IP addresses.

Solution technology definition [a technique] a process to define the platform application environment(s) needed to support and run application(s). 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 progresses from a logical applications architecture through progressively more physical views to 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 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.

9.4: Communication networks and protocols

PAN: **Personal Area Network** [a communication network] typically carried or worn by a person. The reach of a wireless PAN varies from a few centimetres to a few meters.

LAN: **Local Area Network** [a communication network] under the control of a local network administrator, usually within a building or closely connected buildings.

MAN: **Metropolitan Area Network** [a communication network] optimized for a block of buildings, or an entire city, and likely to use leased lines.

WAN: **Wide Area Network** [a communication network] that connects computers and networks over long distances, usually employs leased lines or the Internet.

VPN: **Virtual Private Network** [a communications network] in which communication between nodes is carried by connections within some larger network instead of by physical wires. Usually uses the Internet or other WAN but feels like a LAN. The data link-layer protocols of the virtual network are said to be tunnelled through the wider network.

Cloud computing: an architecture that features services provided by a service provider 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 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 platform services (e.g. message delivery or transaction rollback) from a platform application 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.

Topology [a pattern] for the shape of a network, or communication routes over it. A shape that connects nodes or constrains communications routes over a network.

> **Point-to-point (aka mesh)** [a topology] in which one node sends a message directly to another node; other end-point nodes are unaware of this.
> **Bus** [a topology] in which each node listens to all messages sent by other nodes, and filters out unwanted ones.
> **Hub** [a topology] in which each node sends and receives messages via a central node.
> **Ring** [a topology] in which nodes pass a message around in a circular fashion until it arrives at the intended destination node.

Network layer [a view] a level in a hierarchy of communication layers, corresponding to layers of platform applications.

Protocol [a standard] for the process and rules used by message senders and receivers when they exchange messages via transport mechanisms, or by end points in a telecommunication exchange. May include a standard format for the header preceding the message, the footer following the message, and the sequence in which messages are exchanged.

Protocol Stack [a pattern] that arranges protocols in layers, corresponding to layers of platform applications. The best known protocol stacks are probably OSI and TCP/IP.

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

Application layer [a network layer] the top layer. It provides services to identify communication partners and quality of service, authenticate users, ensure privacy, and identify constraints on data syntax. Popular application layer protocols include FTP (file transfer protocol), HTTP (hypertext transfer protocol), POP3 (post office protocol 3), SMTP (simple mail transfer protocol), also RPC, SOAP, HTTPS, and DNS.

Transport layer [a network layer] that manages the end-to-end message delivery. TCP (transmission control protocol) and UDP (user datagram protocol) have a checksum to ensure data flow integrity. TCP ensures complete data transfer or else times out. By contrast, UDP is a lightweight protocol that does report failed message delivery.

Network layer [a network layer] that handles the routing and forwarding of data at the packet level. It sends outgoing transmissions in the right direction to the right destination. It receives incoming transmissions. E.g. IP is the dominant network layer protocol.

Data transport layer [a network layer] at which data is transported around the physical network modes by network communications software. It may, for example, do bit-stuffing for strings of 1's in excess of 5 etc. E.g. Wi-Fi. Ethernet (based on a bus topology) and Token passing (based on a ring topology).

Physical layer [a network layer] at the bottom, connecting nodes by a physical medium. It conveys the bit stream through the network at the electrical and mechanical level - provides the hardware means of sending and receiving data on a carrier. E.g. 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] a logical identifier, assigned to a node in a network that uses the Internet Protocol for communication. It is divided into two sections: the first identifies a local network; the second identifies a node on that network. An IP4 address is made from four 8-bit parts, often presented as four decimal numbers. An IP6 address is made from eight 16-bit parts, often presented as eight hexadecimal numbers. The range of IP6 addresses is enough to uniquely address every computer, but sub-netting and network address translation remain useful.

NAT: Network address translation [a technique] used to separate the private internet address space within a LAN from the public internet. Most NAT systems enable multiple computers on a private network to access the internet using a single public IP address.

Convergence (of telecommunication media): the trend that enables one operating platform to supply many media. It enables equipment providers to combine voice, data and images in services offered to the user.

VoIP: Voice over IP system [a component] of IP Telephony that is promoted as offering lower network installation and management costs, lower voice phone tariffs and mobility of phone numbers.

9.5 Connecting applications over networks

Process (computer sense): an application (or program instance) running on a computer (as can be seen in the "Task Manager" on a lap top). 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. E.g.

- 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). E.g. 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: A business planning context

CBP: Capability-based planning [a work process] for planning projects or other endeavors to provide capability increments. It provides a context for architecture, but its scope can be much wider. Architecture can inform it, and elaborate on some parts, but is not responsible for all of it.

Capability increment [a work process] a step change in the development of a capability. It may relate to an architecture transition.

10.2: Migration planning

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

Gap analysis (baseline-target) [a technique] for comparing two similar structures, to find items in one that are not in another. 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.

RAID catalogue [an artefact] 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 potential problem; an event that causes an issue if it occurs.
> **Assumption** [an influence] Statement that, if not true, could turn into a risk or issue that threatens the success of a project.
> **Issue** [an influence] a problem that needs resolution. Sometimes the realisation of a pre-identified risk, or an assumption that turned out to be false.
> **Dependency (risk sense)** [an influence] a dependency of a project upon an external actor or deliverable, not under the management of the project manager.

Migration path [an artefact] a progressive series of architecture descriptions, each related to different state of an enterprise or system.

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

Work or product evolution table [an artefact] a table that shows when work or product elements start and stop through a series of transition states.

Roadmap [a document] for some, a plan that adds timescales to a migration path, so sits half-way between a migration path and detailed project plans. (For others, a long-term plan for how an application or technology portfolio will be updated, which may cut across several projects.)

Critical path analysis [a technique] to construct a model of the project that includes (i) a list of all activities required to complete the project (also known as work breakdown structure) (ii) the duration of each activity, and (iii) the dependencies between the activities.

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.

10.3: 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 artefact] 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 [artefact] 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 techniques below.

> **Cost-benefit analysis** [a technique] an assessment of the costs and the benefits of a course of action and/or a proposed system.
>
> **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 description starts in earnest, at several times in the process, and at several levels of design.
>
> **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.

11: Architecture Management

This section addresses the roles and processes needed to govern and implement an architecture description, in development and operation, including the management of changes.

11.1: Architecture Implementation

Architecture implementation [a work process] that realise an architecture description through development and deployment of a system. This requires programme and project management roles and processes. It uses tools for source code management, unit testing, load testing, regression testing, security testing, compliance testing etc.

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 known as Incremental Development in DSDM. It is not necessarily agile.
>
> **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. It features early testing for usability and performance; user involvement and feedback is a prerequisite. It favours 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. Its practices are designed to capitalise on the skills and knowledge of a small team.
>
> It may be seen as the anti-thesis of enterprise architecture. However, a bimodal IT department embraces both. An agile project does need some governance, and there is a role for oversight by a solution architect.

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

> **Transition into operations** [a work process] the production or run-time system is handed over to be run by some kind of managed operations organisation.
>
> **Transition into maintenance** [a work process] the design and compile-time system is handed over to be maintained and perhaps enhanced by some kind of maintenance organisation.

11.2: Architecture Change Management

Architecture change management [a work process] needed to manage changes to architecture descriptions, mostly stemming from changes to requirements or constraints, or operational systems.

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

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] the roles and processes needed within change management to monitor the potential sources of change, record change requests, perform impact analysis and decide which changes should be made.

> **RFC: Request for Change** [an artefact] "Form used to record details of a request for a change to any Configuration Item within an Infrastructure or to procedures and items associated with the Infrastructure." ITIL
>
> **Impact analysis** [a technique] analysis of the effects of a change (perhaps a new requirement or deliverable) to find the effects of that change. How does it impact what has been done so far? How does it constrain what is planned for the future? Leads to 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.

11.3: Architecture Governance

Governance [a work process] That facet of management concerned with ensuring an enterprise does what it is supposed to do - that is, achieves goals, follows rules and delivers what its stakeholders expect. It requires measurement and control of performance. May be subdivided into:

- Corporate governance; the responsibility of the enterprise's executive board.
- IT governance; the responsibility of an IT board.
- Enterprise architecture governance; the responsibility of an architecture board. The management 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, 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. Aka chief architect or design authority.

Architecture compliance review [a work process] for monitoring the compliance of work done to architecture principles, policies and models. Reviews of various kinds may be carried out at various points in the specification and development of a system. Only some of these reviews require a governing architect or use an architecture review checklists.

Architecture review checklist [an artefact] a standard checklist of questions to be asked in 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 description 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, granted by a governing architect, and to be reviewed after the specified time.

Capability maturity model [a reference model] for evaluating the maturity of an organisation and its processes. First and best known is the maturity model is the CMM for software processes, from the Carnegie Mellon University Software Engineering Institute. There are now maturity models for architecture roles and processes, such as those included in the list of references.

11.4: Architecture in Operations

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 focused on end user devices. It may be related to a CMDB.

Appendices: informative but not examinable

This section is not examinable; it is included in the RM 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, object 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 (artefact):** 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 description 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 RM

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

1. The RM 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 RM lists concepts to be understood, not all terms that might be used.
3. The RM 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. RM 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 RM does not invent terms other than is justified by the principle of structural warrant.
6. The RM focuses more on discrete elementary concepts rather than aggregates of them (since the contents of aggregates are more disputable).
7. The RM 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 RM reflects the scope of current architecture frameworks; it does not speculate about future trends.
9. The RM 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. Trim by about 10%. Remove peripheral terms and concepts (which trainers may still choose to teach). 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. Refer back to the Foundation in each relevant section.

5. Although the RM must not recommend any particular standard or source; it should outline the core concepts in ISO 42010.

6. Section 1: Architecture and Architects. Trim back and revise in the light of changes mentioned above.

7. Section 3: Architecture Frameworks: Include entries on abstraction (formerly in section 1), and present the two architecture description classification frameworks as tables.

8. Section 4: Business Architecture. Clarify description of "structured analysis".

9. Section 6: Software Architecture. Revise to put concepts in a clearer historical context. Remove some detail on OO design patterns

10. Section 7: Applications Architecture: 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, to enable more examination questions in this area.

Trademarks