Reference Model for Enterprise and Solution Architecture v16

Thursday, 17 January 2019

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

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

Contents

Introduction to the reference model

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

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

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

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

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

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

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

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

1 Systems, architectures and architects

Enterprise and solution architecture are about the design of business systems in which human and computer actors play roles in processes that create or use business data.

1.1 Foundation (not to be examined)

Architecture
[A work product] that describes a system.
An abstract description of the structural and behavioural elements of a system.
It may map system elements to motivations, constraints
It may map system elements to work packages needed implement the system.

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

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

Architecting
[A work process] that involves

- analysis of the context
- analysing and choosing between options
- defining an architecture (using principles and patterns)
- ensuring agreement of what is described.
- planning the move from the baseline state to the target state
- governing that change.

System (aka activity system)
A system in which components interact in the performance of processes.
A system may be encapsulated behind an interface.
A system may be a subsystem or component of a larger system.
The system boundary is chosen by the observer or designer who describes it.

Business system
[A system] in which processes are designed to achieve the aims of a business.
The system components usually include human and computer actors.
Components interact by creating and interpreting messages.
Components respond to messages in the light of memories retained.
Messages and memories contain data structures that capture information.

System element
A discrete structural or behavioral element of a system.

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

	Behaviours	**Active structures**
External view	**Service contract:** an end-to-end process defined as external entities see it.	**Interface definition**: a declaration of available and accessible behaviours
Internal view	**Process:** a sequence of activities performed by one or more components.	**Component:** a subsystem capable of performing one or more behaviours.

Fig 1.1.

Structural view
[A view] that shows what a system is made of.
It connects system elements (components and objects) as nodes in a structure.

Behavioural view
[A view] that shows what system does over time.
It shows how system elements (processes and services) progress from start to end.

External view
[A view] that shows what an external entity sees of a system or process.
E.g. an interface definition or a service contract.
It may relate system elements to external entities

Internal view
[A view] that shows the internal design of a system or process.
It shows the internal structures or behaviours of a system.
E.g. a network diagram or a process flow chart.

System design principle
A principle for how to approach the specification of a system.
Architecture methods often feature the following principles.

- External before internal
- Behavior before structure
- Business before technology
- Logical before physical.

Structural elements (generic)

Structure
Something a system is made of, an element that is addressable in space.
It may be measured in terms of its size or space occupied.
Active structures perform actions; passive structures are acted on.
(Active structures are sometimes treated passive structures.)

Active structure elements

Component
[An active structure] a subsystem capable of performing one or more behaviours.
Components interact to perform the processes and meet the aims of a larger system.
A component can be replaced by any other component with the same interface(s).

> **Logical component**
> [A component] that specifies what is required of one or more physical components.
> It is more abstract or supplier-independent than a physical component.
> It may specify services to be performed and data resources needed.
> E.g. a logical business, application or technology component.
>
> > **Capability**
> > An ability to do or achieve something.
> > E.g. to perform a process or role, realise function or achieve an aim.
> > In practice, it is often 1 to 1 with a logical component.
>
> **Physical component**
> [A component] that is supplier-specific and/or can realise one or more logical components.
> E.g. a physical business, application or technology component.

Interface definition
[An active structure definition] a declaration of available and accessible behaviours.
It may include both logical interface elements:

- *Signatures* (name, inputs and outputs) for services.
- *I/O flow*s consumed and produced by services.

And physical interface elements:

- *Protocols* used to exchange data between components.
- C*hannels* via which I/O flows are passed.

Passive structure elements

Data object
[A passive structure] in which some meaning or information is encoded.
It can be made, moved or modified.
(NOT an object in the sense of object-oriented programming.)

Location
[A passive structure] a place where active components or passive objects can be found and work can be done.

Behaviour elements (generic)

Behaviour
Something a system does over time.
It either changes the state of a system or reports its current state
It is performed by one or more components.
It may be measured in terms of its start and end times.

Process
[A behavior] comprising one or more activities that run from a start to an end result of value.
E.g. deliver package, check credit, provide weather data, and consolidate reports.
It can be coarse-grained (build a house) or fine-grained (retrieve an address).
It may be defined externally in a service contact.
It may be defined internally as activities under logical control flow.
It may orchestrate subordinate processes.

Service
[A process] regarded as a service by an external entity.
It supports or enables that entity by delivering one or more results.

Service contract
[A behavior definition] as external entities see it, in terms of:

- Entry conditions
- Exit conditions (aka results)
- Qualities.

It encapsulates all processes (human and/or computer) needed to complete the service.
It may be documented along with other service contracts in a service portfolio, a Service-Level Agreement (SLA) or an interface definition.

Entry conditions
Conditions of these kinds may appear in entry conditions.

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

Exit conditions (aka results)
Conditions of these kinds may appear in exit conditions.

> **Output flow:** a material structure or message produced by a successful process.
> **Post condition:** a change of state resulting from a successful process; a change in the qualities of an object that is maintained, including any "added value".
> **Value:** the worth of an output or state change to an owner or consumer.
>
> Note: an I/O flow conveys a material and/or data structures from a sender to a receiver. A material structure of interest is usually accompanied by a data structure.

Qualities
Measurable attributes of a service.
E.g. duration, cost, availability.

1.2 Modularity

The generic metamodel of system elements is repeated below.

	Behaviours	Structures
External view	**Service contract:** an end-to-end process defined as external entities see it.	**Interface definition**: a declaration of available and accessible behaviours
Internal view	**Process:** a sequence of activities performed by one or more components.	**Component:** a subsystem capable of performing one or more behaviours.

Fig 1.1.

System decomposition: [A technique] that divides a larger system into smaller components.
Each component can be seen as a system in its own right and further subdivided.

> **Atomic component:** [A component] not decomposed into smaller components (in a description of a system).

Behavioral decomposition: [A technique] that divides a longer process into shorter activities
Each activity can be seen as process in its own right and further subdivided.

> **Atomic activity:** [A process] not decomposed into shorter processes or activities (in a description of a system).

Encapsulation: [A technique] that defines a thing by an interface it offers.
It hides inner workings or processes from external entities.
It hides internal resources (notably data structures) from external entities.

> **Structural encapsulation:** Encapsulating a component by an interface it offers.
> **Behavioral encapsulation:** Encapsulating a process by a service contract.

Component-based design
[A technique] for defining a system in terms of interacting components.
The requirement for a component is defined as a provided interface definition.
What a component needs is defined as required interface definitions or server component names.

Modularity
Central issues in architectural design include

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

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

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

Note: finer-grained components tend to be more tightly coupled; coarser-grained components tend to be more loosely coupled, so they can be managed and run independently.
Note: one interface definition may encapsulate several components and one service contract may encapsulate several (related) processes. So, it possible to re-engineer (refactor) internal components and processes without changing the external services and interfaces.

1.3 Architecture domains/views

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

Business architecture
[A view] that identifies and relates business elements:
- What the business delivers: business products and services
- How the business does it: business processes (scenarios, value streams)
- What the business needs to do it: components and objects 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:
- Data stores and flows used by business activities
- Data structures contained in the data stores and data flows
- Data qualities: data types, confidentiality, integrity and availability.

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

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

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

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

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

Domain layer	Behaviours	Logical structures	Physical structures
Business	Business services	Business functions, Roles	Organisation units, Actors
Data		Data entities Logical data components	Physical data components
Applications	IS services	Logical application components	Physical application components
Technology	Technology services	Logical technology components	Physical technology components

Fig 1.1. Core concepts in the TOGAF standard v9.2.

1.4 Architect roles

Architect
[A work role] to design, plan and oversee the building of solutions and/or systems.
It involves responding to requests for work, gathering contextual information, producing architectures and governing lower-level design or construction.
Higher level roles abstract from detail and operate more widely, with a broader scope.
Higher level roles may *govern* lower roles to some extent.

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

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

Fig. 1.3 The architecture work space

The rest of this section is limited to short lists, sufficient for examination purposes.

Enterprise architect
[An architect] who takes a cross-organisational and strategic view of business systems, looking to.

- optimise business systems by digitisation, standardisation and integration
- exploit business data captured by business processes
- identify potential innovations in business processes
- maintain enterprise-wide business, data, application and technology component portfolios
- maintain cross-organisational and/or strategic road maps
- shape, steer and govern the work of solution architects.

Solution architect
[An architect] who focuses on individual solutions and systems.

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

Software architect
[An architect] who addresses the fine-grained modularisation and integration of software components inside business applications.
He/she may follow principles and patterns for modularisation and integration.

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

1.5 Architect goals and skills

This section is deliberately limited to short lists, sufficient for examination purposes.

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

Solution architect goal
[An aim] E.g.
- Support the goals (above) of enterprise architects
- Alignment of IS/IT to business processes and roles
- Quality of IS/IT project deliverables.
- Cost of IS/IT project deliverables (though a manager is usually *accountable*)
- Timeliness of IS/IT project deliverables (though a manager is usually *accountable*)
- Solution-level risk identification and mitigation.
- Application integration and data integrity.
- Conformance of solutions to non-functional and audit requirements.
- Conformance of solutions to principles, standards, legislation.
- Effective cooperation between managers and technicians.
- Governance of detailed design to architecture principles and standards.

Architect knowledge or skill
[A property] 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.

2: Context and motivations (precursors)

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

2.1 Foundation (not to be examined)

Architecture context
The setting of and influences upon a system.
It includes what is known of the wider environment that architects should be aware of.
E.g. stakeholders, systems, concerns, influences and information.

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

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

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

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

2.2 Stakeholders and concerns

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

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

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

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

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

Sponsor
[A stakeholder] 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 Requirements and constraints

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

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

> **Principle**
> [A directive] that is strategic and not-directly-actionable.
> (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 embeds a policy in business system operations.
> (E.g. Access Level = Low if User Type = Public.)

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

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

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

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

2.4 Requirement types

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

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

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

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

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

> **Audit requirement**
> [A requirement type] how an auditor can find the when/where/how/who of activities performed and data recorded, and replay events.
> It may have implications for data records and retention.
>
> **Regulatory requirement**
> [A requirement type] including legislation and regulations that direct or constrain architecture work.
> - Data protection: E.g. GDPR.
> - Data freedom: E.g. Freedom of Information Act 2000.
> - Disability and accessibility: E.g. UK Equality act. US Americans with Disability act. W3C Web Content Accessibility Guidelines.
> - Shareholder protection and audit: E.g. US Sarbanes-Oxley act 2002. Basel I/II/III.
> - Health and Safety.
> - Intellectual property rights:

2.5 Scoping deliverables and artifacts

Solution architecture deliverable
[An architecture] that describes a solution to problems and requirements.
It describes one or more systems and their components.
It may map elements to directives, aims and plans.
It may be defined at several levels of abstraction.

> **Solution concept/vision**
> [An architecture] that is the first response to a business problem or vision.
> It briefly describes a target, just enough to enable options to be compared.
> It includes an initial idea of benefits, costs and risks
> It outlines architecture work to follow.
>
> **Solution outline**
> [An architecture] that describes the high-level design of a target system.
> It is produced after a first pass architecture.
> It refines estimates of cost, benefits and risks, and further work to be done.
>
> **High level design**
> [An architecture] that is sometimes a synonym for a solution outline.
> Sometimes an extra level between solution outline and detailed design
>
> **Detailed design (or low level design)**
> [An architecture] that describes a project-ready architecture.
> It is complete enough for the project to be scheduled, resourced and started.
> The building team will refine the detailed design.

Scope dimension
[A view] a dimension of scope.

- Breadth: scope of the enterprise, system or solution (see overview 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.

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

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

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

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

2.6 Other contextual terms

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

> *A service contact* declares the parameters any client uses to request one service.
> The service requester, performer and consumer may be three different parties.
>
> *A collaboration agreement* covers all services one client can request from one server.
> It may be built by selection and aggregation of service contracts.
> It may list only the exit conditions or results of services.
> It may do no more than name some services results.

Measure
A measure of a system/component or service/process.
A quantity set as a target goal or requirement, or measured/reported in system operation.
The measure type is typically a non-functional quality, requirement or attribute.
E.g. cost, size, duration, throughput or confidentiality.
A measure instance is a target or actual level of a measure.

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

3 Architecture frameworks

3.1 Foundation (not to be examined)

Architecture framework
A comprehensive architecture framework contains advice on:
- Processes - for architecting
- Products - for architecture
- People – architect and related roles.

TOGAF®: The Open Group Architecture Framework
[An architecture framework] published on a free-to-read public web site.
Its use by a commercial organisation is restricted by copyright conditions.
It is a relatively abstract management framework for architecture work.
It is centred on a process called the architecture development method.
Originally focused on technology architecture; now focused more on business architecture.
Originally intended for transforming an enterprise's business systems from a baseline state to a target state; now commonly used at a narrower and more tactical level.

AM: Avancier Methods
[An architecture framework] published on a free-to-read public web site.
Its use by a commercial organisation is restricted by copyright conditions
It provides relatively concrete guidance for solution architecture work.
It does also address enterprise portfolio management and rationalisation.

3.2 Architecture processes

ADM: Architecture Development Method
[The process] defined in TOGAF to develop and use an enterprise architecture.
Users are expected to adapt the process and iterate through it.
It is centered on a cycle of 8 phases.
- A: Architecture Vision
- B: Business Architecture
- C: Information System Architecture (Data and Applications)
- D: Technology Architecture
- E: Opportunities and Solutions
- F: Migration Planning
- G: Implementation Governance
- H: Architecture Change Management.

Avancier Methods (AM) process
[The process] defined in AM, primarily to develop and use a solution architecture.
There are four broad phases: Initiate, Architect, Plan and Govern.
Each phase is subdivided into lower level processes.
Users are expected to adapt the process and iterate through it.

3.3 Architecture products

Architecture content framework
[A passive structure] for organising the contents of architectures, typically composed of deliverables, artifacts and entities.

> **Architecture deliverable**
> [A document] that architects produce or contribute to.
> It is typically circulated for approval by sponsors if not all stakeholders.
> It should conform to a standard document type. It often contains artifacts.
>
> **Architecture artifact**
> [A model] that typically takes the form of a catalogue, matrix or diagram.
> It is composed by relating architectural entities.
> It may appear in one or more deliverables.
>
> **Architecture entity**
> [An entity] that appears in one or more artifacts.
> E.g. Process, Organisation, Location, Data entity, Application, Technology.

Architecture repository
[A data store] an information base used by architects; a database that holds descriptions of an enterprise's business and its systems, and the meta data that describes those descriptions.
Its structure is defined in some kind of schema or architecture meta model.
The content of the repository can be categorised in many ways.
For example, using the Zachman Framework or the Enterprise Continuum.

Mapping
[A correspondence] drawn between architectural entities in an architecture repository.
Mappings may be made the purposes of:

- gap analysis (see section 10)
- impact analysis (see section 11)
- requirements traceability analysis (see section 2)
- cluster analysis (see section 4).

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

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

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

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

3.4 Architecture models

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

Model kind
[A work product description] that typifies a model and provides a template for building a model.

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] for systems and software engineering, originally funded by the US DoD.
> Its includes 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.
> Initially designed to help in OO software design, it is now used outside of that.
> It includes structural models such as class diagrams and deployment diagrams.
> It includes behavioural models such as use case, activity and sequence diagrams.
>
> **ArchiMate**
> [A modelling language] maintained by the Open Group.
> Components, interfaces and services are shown in distinct boxes.
> It overlaps with UML, but is intended for more abstract architectural design.

MDE: Model-Driven Engineering
[A technique] used in methods and tools for forward engineering and reverse engineering, that is, the process of transforming a conceptual model to a logical model to a physical model, and the reverse of that process.

> **Model-Driven Architecture (MDA)**
> [A model-driven engineering technique] a vision of the Object Management Group (OMG) that encourages suppliers to develop tools 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.

3.5 Abstraction techniques

There two broad kinds of abstraction.
1. Abstraction from reality, creating a description, the opposite of *concretion.*
2. Abstraction from detail, creating a simpler description, the opposite of *refinement.*

Architects use a mixture of abstraction techniques to hide details and assist explanations.
Techniques include omission, summarisation, delegation, generalisation and idealisation.

Abstraction by delegation (from server to client)
[A technique] that simplifies clients by hiding work they delegate to servers.
A *client* requests a service from a server; a *server* performs services requested by a client.

> **Layered client-server hierarchy**
> See section 1.

Abstraction by composition (from smaller to larger)
[A technique] that simplifies by hiding small things ones inside larger ones.
A coarse-grained description features only large system elements.
A fine-grained description features shows more detail by way of smaller system elements.

> **Hierarchical decomposition**
> See section 1.

Abstraction by generalisation (from unique to universal)
[A technique] that simplifies by hiding differences between things.
A *generic* description is more widely applicable.
A *specific* description is more narrowly applicable; it may extend a more generic description with additional properties.

> **Class hierarchy**
> A hierarchy in which a generic description is incrementally specialised into successively more specific descriptions.

Abstraction by idealisation (from reality to concept)
[A technique] that simplifies by hiding physical and/or supplier-specific details.
The classic idealisation hierarchy "reverse engineers" from real to conceptual.

> **Conceptual (or domain) model**
> [A model] that defines terms and concepts in a business or problem domain without reference to any computer or software application.
>
> **Logical model**
> [A model] that excludes details of that system's physical implementation. It is supplier-independent and portable.
> It may specify services or processes to be performed, and data or abilities needed.
> It leaves open the choice of particular components, products and mechanisms.
>
> **Physical model**
> [A model] that is supplier-specific or includes implementation details.
> A description of particular products or mechanisms that can be employed or deployed to realise a logical model.

3.6 Pre-defined classifications and reference models

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

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

Fig. 3.1 The Zachman Framework

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

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

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

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

Fig. 3.2 The Enterprise Continuum

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

4: Business domain view

4.1 Foundation (not to be examined)

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

Fig. 4.1a Base business architecture concepts

Enterprise
[A system] a human-led organisation with shared goals and budget.
It is usually the highest level, spanning several organisation units.
It may be a segment of an organisation.
It may be in the public or private sector.

Business domain
A class of organisation, or segment thereof, based on the services it offers.
(E.g. law, employment law, telesales, insurance, airline operation, airline maintenance, security, emergency response.)

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

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

Business model
[A description] of how an enterprise delivers value to its owners or sponsors.
It may take the form of a top-level summary diagram or document.

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 is presented as two-by-two grid of process standardisation against integration.

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

Fig. 4.1b Strategic coordination view (“EA as Strategy” by Ross, Weill and Robertson).

4.2 Business structure concepts

Business component
A division of a business; a node in a structural view of business operations.
Core divisions develop, market, sell and deliver business-specific products and services.
Support divisions (such as personnel or facilities management) are similar in different businesses, and are therefore candidates to be out-sourced.

> **Business function**
> A logical business component.
> It typically groups activities with one higher level aim or cohesive aims.
>
> **Capability**
> The ability to perform some activity or achieve some aim.
> Often, a "business capability" is the ability to realise a "business function".
> However, a capability (e.g. skill development) may cross business functions.
>
> **Organisation unit**
> A physical business component.
> It typically groups roles or actors into a manageable group
> It is expected to have goals and objectives, a manager, and a budget.
>
> **Role**
> A logical business component
> It groups behaviours performable by an actor.
> E.g. loan applicant, expense claimant, expense claim approver.
> It may define abilities required to play the role.
>
> **Actor**
> A physical entity able to play one or more roles.
> The entity may be person, an organisation, a natural, mechanical or information system.

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

4.3 Business structure decomposition

[A technique] that successively divides an enterprise into smaller divisions.
It is used a basis for business analysis, heat mapping and classification of other architectural entities.

Functional decomposition: A hierarchy of functions.

> **Functional decomposition diagram**
> A diagram that relates business functions to each other in a logical organization structure.
> It decomposes the topmost functions into smaller functions.
> It typically stops at a 3rd or 4th level of decomposition, though this varies.
> It should be stable and ideally contains no duplicate elements.
> It is commonly used as a base artifact for enterprise architecture, especially where the organization's physical management structure is unstable and/or not "functional".
>
> Purposes: To show what a business does regardless of who does it (management structure) and how it's done (process flows).
> To name, classify and structure business activity domains that are supportable by IT.
> To enable heat-mapping (color coding) to show where changes are needed or proposed, or priorities or phases in a change program.
> To provide a structure for cataloguing other architecture entities.
>
> **Business capability map/diagram**
> A diagram that shows the logical capabilities that a business needs to meet its purposes.
> Most if not all examples correspond to a functional decomposition diagram.
> More generally, a capability (e.g. skill development) may cross business functions.

Organisation decomposition: A structure of organisation units, which stops at the level of human actors. It usually shows reporting lines between unit managers.

> **Organisation decomposition diagram**
> A diagram that shows the (physical) management structure of an organisation.
> Typically, the organisation units are hierarchically decomposed
> The diagram stops short of the individual actors employed.
> Purposes: to help architects identify/show where business functions/capabilities are or should be realised.

Atomic business component
An elementary business component, not further subdivided.
E.g. A node at the bottom of a functional decomposition (a group of atomic activities).
Or a unit at the bottom of an organisation decomposition (a group of actors).

4.4 Other business structure artifacts and techniques

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

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

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

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

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

Business data model
See the Data domain section.

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

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

4.5 Business behaviour concepts and artifacts

Business behaviour
A repeatable activity that (if successful) produces result(s) of value to the consumer of those result(s).

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

> **Business service contract**
> (See the general service contract in section 1.)

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

> **Value chain diagram**
> [An artifact] that indicates how business segments or functions deliver value.
> A top-level view that divides business into core and support segments or functions.
> It suggests a flow of materials and/or data between core segments or functions.
> It may show relationships to entities outside of the enterprise.
> It shows on one page how core business segments or functions relate to each other and generate "value".
>
> **Process diagram**
> [An artifact] that represents activities in a process from start to end.
> The process may complete within a function or cross function boundaries.
> Initially, architects focus on what is called the main, straight-thru, or happy path.
> However 80% of the complexity lies in the 20% of exception paths.
>
> > **Value stream diagram**
> > [A process diagram] that shows the end-to-end stages in a business process..
> > It may decompose each stage into a list of activities.
> > The control logic governing activities is usually not shown.
> > It may show the exit conditions of the process.
> > Including a result of value to a customer, stakeholder or end user
> >
> > **Business scenario diagram**
> > [A process diagram] that shows how a process meets a business aim.
> > It shows also roles played by human and computer actors in each process step.
> > It is a common way to identify and clarify architecture requirements.
> >
> > **Process flow diagram**
> > [A process diagram] that shows the control logic of activities in a process.
> > It relates the stage or steps of a process in a sequence that yields results of value.
> > It may show business components involved in performing process steps.
> > It may include exception paths.

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

4.6 Business behaviour decomposition

A technique that successively divides longer behaviours into shorter ones.
Architects use this both to capture requirements and design business operations.

Service decomposition
Showing one service as dependent on subordinate services.

Process decomposition
Showing one step/activity in a longer process as a lower-level process of several activities.

Atomic business behaviour
An elementary behaviour, not further subdivided.

Atomic business service
A business service performed by an atomic business component.

Atomic business activity
A process step or activity at the bottom of a business process decomposition.
It may be at the one person, one place, one time (OPOPOT) level.
It may be more coarse-grained – more than one person over an extended time.

Structured analysis verification
[A technique] to ensure a business architecture is comprehensive and consistent.
A structural decomposition usually stops at a 3rd or 4th level.
Activities people perform may be at a much finer-grained, say 6th or 7th level.
Every atomic activity found in a business process model should be locatable under one or more business functions.
Conversely, every atomic business function/capability may be mapped to one or more stages in a high level value stream, scenario or process (as illustrated in BizBOK).

Process automation
[A technique] that decomposes a business processes into steps and identifies application use cases needed at each step.
Use cases may be further decomposed to identify automated behaviors needed.

5 Data domain view

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

5.1 Foundation (not to be 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

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

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

Structured data
[A data object] that fits a pre-defined data type or structure of data items.
It usually records real word entities or events.
It may contain references to unstructured data.

Unstructured data
[A data object] containing text or images that do not fit a pre-defined data type or structure.
E.g. emails, voice and video.
However, it may contain recognisable structured items.

Meta data
[A description] of data.
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.

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.

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

5.2 Data at rest (physical)

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

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

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

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

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

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

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

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

Big data schema
[A data store] characterised by a high volume of data, high velocity of data capture and a wide variety of data.
Big data store features include:
Search and discovery: gathers information from diverse data sources.
Map and reduce: compresses a data structure into a set of key/value pairs (aka tuples).
Sharding: divides the population of a data entity type across (potentially thousands of) physically data servers.
Advanced analytics: identifies patterns in data.

5.3 Data at rest (logical)

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

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

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

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

5.4 Data at rest (conceptual)

Business data model
[A data structure] composed of data entities recognised by business people.
It may be a very abstract model of data stored across a whole enterprise.
Or a more detailed model of data in several data stores in one business domain.
Some maintain a business data entity catalogue instead of a data model.

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

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

5.5 Data in motion (in flow)

Data flow
[A motion] the passage of data structure in a message, file, form, report, display from a sender to a receiver.

Data flow or message catalogue
[An artefact] that lists the data flows transported from senders to receivers, with attributes such as trigger, sender, receiver, transport technology and non-functional measures

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

> **Logical data flow structure (or regular expression)**
> [A data flow structure] that is a hierarchy in which every element is part of a sequence, or an option of a selection or an occurrence of an iteration.
>
> **Physical data flow structure**
> [A data flow structure] represented in the format or schema required by a particular data format or technology.

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

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

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

5.6 Data qualities and integration

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

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

- Consistent: a data item (e.g. customer name) has the same value in every part of a distributed system, in all locations that data item is stored.
- Conformant: a data item obeys relevant business rules, sometimes in relation to another data item. 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.

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 view

Applications architecture (section 7) depends on the lower level design of fine-grained software components discussed in this section. Enterprise and solution architects should be aware of tools and techniques for modularising one application into components and integrating those components.

6.1 Foundation (not to be examined)

	Required behaviours	Logical structures	Physical structures
Software	Operation	API	Application component

Fig. 6.1 Base software architecture concepts

Component-based design (software level)
[A technique] a modular design approach that divides a system into components.
Components may be coded using different technologies and deployed in different locations.

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

Operation
[An automated behavior] that can be requested of an application component.

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

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

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

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

Artifacts

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

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

6.2 Component integration

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

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

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

OO Design
[A technique] for modular design and integration that initially assumed the run-time system works like this :

- Instantiation: an object is an instance of a class (a component type)
- Identification: an object is identified by an object identifier
- Co-location: client and server objects work in the same name space
- Statefulness: an object is stateful
- Inheritance: a subtype object can perform the behaviors of a super type class
- Design pattern: intelligent domain objects communicate to complete a process
- Synchronicity: client objects make request-reply invocations to server objects
- Blocking: a server object accepts only one invocation at once.

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

IDL: Interface Definition Language
A language for defining an API (not the procedures of operations/services in it).
It enables components coded in different languages and running on different operating systems to interoperate.

WSDL: Web Service Description Language:
[A standard] from the W3C for an interface definition.
A WSDL includes a signature, protocol and web address for each operation/service.
Initially it depended on XML and SOAP, now usable with JSON and HTTP.

SOAP
A standard application layer protocol devised by Microsoft, then adopted by the W3C.
A protocol used to invoke operations on remote components without using an ORB.
It allows distributed components to communicate by sending XML messages over HTTP.

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

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

OData
[An IDL] an evolution of REST.
It supports clients wanting to invoke operations on entities in a remote web data store.
Client applications that speak OData can easily connect to data server applications that provide CRUD operations on a logical data model.

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

6.3 Component coupling

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

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

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

Factor	Tight coupling	Decoupling techniques
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 design patterns [not to be examined]

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

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

Model View Controller
[A pattern] that separates

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

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

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

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

6.5 Communication patterns

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

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

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

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

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

Aside: Different communication styles may be used at different levels of a communication stack. Under the covers is always some point-to-point communication.

7 Applications domain view

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

7.1 Foundation (not to be examined)

Required behaviours	Logical structures	Physical structures
Application service	Application interface definition	Application

Fig. 7.1 Base applications architecture concepts.

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

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

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

> **Logical application component**
> [An application component] specified by services offered and data maintained.
> The specification is independent of implementation and technology.
> It is typically mapped to one physical application component.
> But the relationship may be more complex.
>
> **Physical application component**
> [An application component] that realises a logical application component specification.
> It uses particular technologies.
> It typically realises one logical application component.
> But the relationship may be more complex.
>
> **Microservice**
> [An application component] encapsulated behind an API, that is large enough to be regarded as an application, but small enough to suit agile development and enhancement.
> It is usually a subdivision of what could be a larger application, and maintains a subset of what could be a larger database.
> Data integrity is maintained using messaging passing rather than ACID transaction roll back.
> So, using the BASE principle, designers must analyse the consequences of temporary data integrity and consider compensating transactions to restore it.

7.2 Application portfolio management

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

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

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

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

Classification by 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 may maintain data for Manufacturing, Supply Chain Management, Financials, Projects, Human Resources, and other functions listed below.
>
> **Customer relationship management (CRM), Call Centre & Marketing**
> [A business application] that supports the development and maintenance of mutually beneficial long-term relationships with customers.
> It helps with attracting customers, transacting business with customers, servicing and supporting customer, enhancing customer relationships.
>
> **Other common business application functions**
> - Accounting
> - Financial Reporting
> - Data Warehousing, Business Intelligence and CPM.
> - Document Management, Content Management an BPM.
> - HR and Payroll
> - Project Management

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

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

7.3 Application behaviour

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

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

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

> **Automated business service**
> [An automated behavior] whose input and output data is defined in a canonical data model.
>
> **Automated data service**
> [An automated behavior] whose input and output data items are defined according to the parochial or physical data model of a specific data source.

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

> **ACID transaction**
> [A transaction] that is Atomic, Consistent, Isolated and Durable.
> It can be rolled back if a specified precondition is violated.
> Using a transaction manager to automate the roll back of a transaction preserves the integrity of stored data and simplifies design and development.
> But is often impossible in loosely-coupled and distributed systems.
>
> **Compensating transaction**
> [A transaction] to handle the side effects of a process (or workflow) that started but could not complete successfully.
> It may undo updates committed to databases, remove messages placed in message queues, send follow-up correction messages, or report cases of data disintegrity.

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

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

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

7.4 Basic design pattern pairs

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

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.

Structural patterns

Hierarchical structure:
[A design pattern] that divides a node into subordinate nodes.
A rule of thumb for division: divide one into about seven.
A rule of thumb for decomposition: stop at three or four levels.
Network structure:
[A design pattern] in which a node can be connected to any other node.

Communication path patterns

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

Delegation patterns

Hierarchical (or layered):
[A design pattern] in which higher components delegate work to lower (server) components**.**
This pattern is widely used to structure complicated systems (machines, networks, software applications and enterprise architecture).
Peer-to-peer:
[A design pattern] in which any two components can delegate to work to each other.
Such cyclic dependencies are said to create a fragile structure, difficult to understand and maintain, but are sometimes inevitable.

Process control patterns

Fork/Orchestration:
[A design pattern] in which one component schedules other components.
It centralises intelligence about the overall process or workflow.
Chain/Choreography:
[A design pattern] in which components interact directly.
It distributes intelligence about the overall processes
Each component calls the next component.

7.5 Applications integration patterns and tools (not to be examined)

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

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

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

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

Point to Point messaging tool
See “RPC” and “ORB” in section 6.

Web Service tool
See “Web Services” in section 6.

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

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

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

7.6 Applications integration patterns (not to be examined)

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

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

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

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

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

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

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

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

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

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

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

8 Design for qualities

8.1 Non-functional requirement types

Performance
[A requirement type] 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.
Measures usually refer to availability at the primary site, excluding disasters.

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

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

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

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

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

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

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

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

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

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

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

8.2 Design for speed or response time

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

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

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

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

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

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

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

Throttling of throughput
To prevent servers being overloaded.

8.3 Design for throughput or capacity

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

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

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

8.4 Design for availability (reliability and recoverability)

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

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

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

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

8.5 Design for security – business-oriented concerns

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

Design for 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.

8.6 Design for security – data-oriented concerns

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

Data security
[A concern] that may be defined as **c**onfidentiality alone.
Or as 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.

8.7 Design for security – application-oriented concerns

Design to protect business applications from unauthorised or malicious use.

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

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

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

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

It is usually followed by authorisation.

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

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

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

8.8 Design for security – technology-oriented concerns

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

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

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

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

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

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

8.9 Other techniques

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

Design for serviceability
[A technique] to monitor servers, networks and instrument applications so that they report on what they are doing, and how well they are doing it.

9 Technology domain view

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

9.1 Foundation (not to be examined)

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

Fig. 9.1 Base infrastructure architecture concepts

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

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

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

> **Logical technology component**
> [A technology component] that is definable by services offered, independently of implementation and technology. For example, the specification of a database or message broker technology to support and enable applications. It is typically mapped to one physical technology component, but the relationship may be more complex.
>
> **Physical technology component**
> [A technology component] that is realised using a particular technology product, which may be hired or bought. It typically realises one logical technology component, but the relationship may be more complex.
>
> **Server node**
> [A technology component] that is a computational resource upon which components may be deployed for execution. Nodes can be connected a network structure or topology. Nodes may be virtual or physical. Nodes may be nested.

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

9.2 Enterprise technology rationalisation

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

ETC: Enterprise Technology Classification
[A passive structure] for 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 technology component] that enables 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 technology component] that enables programs to store and retrieve persistent data.
> (E.g. DB2, SQL Server, Oracle, Sybase and Teradata.)
>
> **Middleware** (See section 7.)

TRM: Technical reference model
[A passive structure] for a technology services catalogue.
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 components and defining a de-duplicated target architecture. The basis of TOGAF versions 1 to 7.

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

Virtual machine
[A technology component] that enables application programs to run above – decoupled from - the underlying operating system and/or hardware processor.
It enables applications to run on different operating systems and/or processors
It can be used in server consolidation.

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

- **Software as a Service**: provision of use cases (e.g. order capture and payment validation) from a business application owned by a service provider; the customer owns only the data.
- **Platform as a Service**: provision of technology services (e.g. message delivery or transaction rollback) by technology components owned by a service provider; the customer owns the business application as well as the data.
- **Infrastructure as a Service**: provision of basic computing technology services, processor speed and memory, by a remote service provider.

9.3: Solution technology elements and definition

Device
[A node] with processing capability upon which components 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 components.
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 component (or artifact)
A software deliverable (source file, script, executable, database table etc.) that can be deployed along with its descriptor to a node. One component can deployed to another.

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

1. Identify the context and requirements for the platform technologies
2. Establish baseline opportunities and constraints
3. Define client nodes and data server or source nodes (show stoppers)
4. Define intermediate web and app 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.

Artifacts

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

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

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

Communications engineering diagram
[An artifact] that shows devices, and is focused on detailing the network components, sometimes annotated with IP addresses.

9.4: Communication networks and protocols

Network scope

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
It likely uses leased lines.

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

VPN: Virtual Private Network
[A communications network] carried by connections within a larger network.
It feels like a LAN but uses the Internet or other WAN.
Its data link-layer protocols are said to be tunnelled through the wider network.

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.

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.
It may corresponds to a specific platform technology.

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. It may standardise format for the header preceding the message, the footer following the message, and the sequence in which messages are exchanged.

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

Application layer: provides services to identify communication partners and quality of service, authenticate users, ensure privacy, and identify constraints on data syntax.
Popular protocols include FTP , HTTP, POP3, SMTP, RPC, SOAP and DNS.

Transport layer: manages the end-to-end message delivery, using a checksum to ensure data flow integrity.
UDP (user datagram protocol) is lightweight, does not report failed message delivery.
TCP (transmission control protocol) ensures delivery or else times out.

Network layer: 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: transports data 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: the physical medium connecting nodes.
It conveys the bit stream at the electrical and mechanical level.
It 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] logical identifies a node in a network that uses the Internet Protocol.
The first section identifies a local network; the second identifies a node on that network.
An IP4 address has four 8-bit parts, often presented as four decimal numbers.
An IP6 address has 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] that enables computers on a private network to access the internet using a single public IP address, by separating the address space within a LAN from the public internet.

Convergence (of telecommunication media)
The trend for one operating platform to supply many media, which enables equipment providers to combine data, voice and video services.
For example, a Voice over IP system (VOIP) for IP Telephony is offers 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. 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 Migration planning concepts

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

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

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

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

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

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

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

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

10.2 Business cases

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

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

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

> **Cost-benefit analysis**
> [A technique] to assess the costs and the benefits of a course of action and/or a proposed system.
>
> **Risk analysis** (see section 10.1)
>
> **Gap analysis (options)**
> [A technique] for comparing two similar structures, to find items in one that are not in another.
> It is used in a business cases to compare optional solutions.
> It helps if the options are presented under the same structure as each other, or with reference to a more general structure.
>
> **Architecture trade-off analysis**
> [A technique] for comparing system options and trade offs between them with a view to selecting one option.
> It may employ a technique such as a Pugh Matrx.

10.3 General planning concepts (not to be examined)

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

> **Programme**
> [A work process] a set of projects that are related by a common goal or shared budget, usually under one manager.
>
> **Project**
> [A work process] with defined start and finish criteria undertaken to create a product or service in accordance with specified resources and requirements.
> Given a time span and a budget, it uses resources to deliver a required outcome, usually under one manager.
>
> **Work package**
> [A work process] a subset of a project's work breakdown structure, defined to yield defined deliverables.
> May itself be decomposed into tasks assigned to different project roles.

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

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

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

> **Risk**
> [An influence] a variation from what is expected or assumed.
> It is usually a potential problem; an event that causes an issue if it occurs.
>
> **Assumption**
> [An influence] a belief or understanding that, if not true, could turn into a risk or issue.
>
> **Issue**
> [An influence] a problem that needs resolution.
> It may be the realisation of a pre-identified risk.
> It may be an assumption that turned out to be false.
>
> **Dependency (risk sense)**
> [An influence] a dependency upon an external actor or deliverable that is not under the management of the programme or project manager.

11: Architecture management

11.1 Architecture Implementation

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

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

> **Waterfall**
> [A technique] a development process that places analysis, design, build, test and roll out in sequence. Engineers proceed from one kind of work to the next without significant iteration or parallelism between stages.
>
> **Iterative Development**
> [A technique] a development process that proceeds by increments, meaning that a working subset of the full solution is delivered as early as possible.
> It is a foundation of the Unified Method and known as Incremental Development in DSDM.
> It is an essential feature of agile methods, and may be used in non-agile projects also.
>
> **Agile Development**
> [A technique] a solution development process that is not only iterative, but also flexible about the requirements, the solution and the process being followed.
> It favours negotiation over planning, and flexibility about requirements.
> 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.
>
> In some ways, agile development is the anti-thesis of enterprise architecture.
> However, both approaches are used in a "bimodal" IT department.
> Once a solution outline has been approved, agile development methods may be used.
> Architects may govern the acceptability of changes during an agile project.
> Accepted changes should be reflected in architecture-level documentation.

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

> **Transition into business operations**
> [A work process] that hands over a completed solution (including any production system) to a business unit for business process operation and management.
>
> **Transition into IT operations**
> [A work process] the hands over a production system to be run by some kind of managed operations organisation.
>
> **Transition into maintenance**
> [A work process] that hands over design and compile-time system to be maintained and perhaps enhanced by some kind of maintenance organisation.

11.2 Architecture governance

Governance
[A discipline] for monitoring and steering the management of an enterprise in accord with overarching drivers, goals/objectives and principles/policies.
It may be subdivided into:

- Corporate governance: the responsibility of the enterprise's executive board.
- IT governance: the responsibility of an IT board.
- Architecture governance: the responsibility of an architecture board.

Different enterprise relate these governance organisations in different ways.

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

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

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

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

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

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

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

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

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

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

11.3 Architecture change management (not to be examined)

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

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 artifact] "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.

11.4 Architecture in operations [not to be examined, MAY BE DROPPED?]

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

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

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

IT4IT
A product of The Open Group that applies enterprise architecture principles to ITSM.
It defines a value chain with four primary value streams.
It decomposes each value stream into functions and defines artifacts that are produced by and exchanged between functions.

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

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

Appendices: informative but not examinable

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

A taxonomy of architecture work

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

A taxonomy of description concepts

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

Generic architecture concepts

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

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

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

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

Relations between concepts

One model kind can be manifest in many models. E.g. a process flow chart notation is a model kind. A process flow chart of cooking an egg (showing activities under a control flow) is a model.
For one system, *one viewpoint is manifested in one and only one view.* E.g. in a process flow viewpoint:

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

One process flow view will contain *all* the process flow charts drawn for one system.
One view can contain several models of several kinds. E.g. in a business viewpoint:

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

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

Principles followed in the reference model

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

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

Change history

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

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

Trademarks