Table of Contents for Chapter 5

Part 1. Architecture 101 – 'The Language We Speak'

<<< … >>>

Chapter 5. Architecture Puzzle

In this chapter, we define *Software Architecture* and *Enterprise Architecture* – something that we, *IT Architects* of various kinds and shapes, do. Yes, we too could not resist. We have to give a definition of what we preach and practise. As soon as we able to define what we do, we can start thinking about how to be good at it.

Sometimes we use *IT Architecture* and *IT Architect* as more generic labels for many other overlapping and related notions, in order to avoid still present fuzziness of definitions in the *Software Architecture* field. In this chapter, we may just use terms *Architecture* and *Architect* meaning all of the above, unless ambiguity in discussion requires greater specificity.

Architecture may be many things to many people. It seems everyone has an intuitive understanding of the term *Architecture* as it applies to the Information Technology. Anyone can give own definition of the *Architecture* that, upon closer inspection and careful qualification, almost guaranteed to be 'good enough'. And, there are many existing adopted definitions of *Architecture* with its views and layers to choose from.

Merriam-Webster's Dictionary gives us as good starting point as any: "Architecture – the manner in which components of a computer or computer system are organized and integrated".

What is *Architecture*?

In the most general sense, the *Architecture* of the system consists of some parts, structure of these parts, externally visible properties and interfaces of parts, and the relationships and constraints between parts. Parts of the *Architecture* may represent some components, partitions and layers that can be de-coupled from the rest of the bigger picture, and viewed separately of the surrounding context.
We then analyse every component on every level or layer in order to understand further:

- Visible external ***Properties*** of the *Architecture* component
- ***Behaviours*** of the component
- ***Events***, exceptions or signals that component is able to consume or emit while playing its role of being the part of bigger *Architecture*
- ***Interfaces*** with the surrounding context (ie. other components and layers in the *Architecture*) or with the wide wild external world. Actually, Interface definition comprises abovementioned points (properties, behaviours, and events). We make a separate point out of Interface due to the importance of this notion. In a well-designed *Architecture*, component ***is*** its public interface for all intents and purposes
- ***Processes*** that the component participates in, interacting with some other components and layers. Architecture should identify processes for every stage in the component's life cycle, and every significant use in fulfilling the business goals of the Enterprise
- *Internal* ***Structure*** of the component or layer aiming to implement the required interfaces of the component, with required quality.

This definition of *Architecture* appears to be generic and recursive, i.e. applicable to any system, and to any part of the system on various levels of granularity and refinement. Despite having this vision as a commonly accepted starting point, methodologies and approaches may vary dramatically, at least on the surface.

Note that the *Internal Structure* of the component was mentioned last, arguably as the point of the lesser importance from the *Architecture* prospective. This is not to say that Architecture does not care about component's internals: one component's internals – another component's external context.
Here lies the fundamental quality of the good Architecture – re-use and repleceability of implementation of the component, for the ease of integration and for containment of the impact of component's evolution on its context in the Architecture.

Architecture schools and authors differ in their ways of how they slice and dice the complex system models further. In dissecting the system, different methodologies may define their own orthogonal dimensions or views, constructs and principles, and own approaches in pulling apart the components and layers of the larger system.

No single view or level of refinement of the *Architecture* will present a sufficiently comprehensive model of the system – only multitude of views, depending on the complexity of the system and fit for purpose.

There is no size for the definition and documentation of the *Architecture* that fits all.
In other words, there are many ways to skin a cat… (Apologies to the animal lovers, just a figure of speech. I did not really mean to do this to my, or to any other cat. Usually, no animal gets hurt in the process of designing the *Architecture*. But then again, mind of the *Software Architect* could be a mystery…)

Levels of Abstraction and Granularity

Human brain has natural limits to complexity that it can deal with. Our ability limited by how many entities and relationships we can grasp simultaneously at a time (seven is an indicative number).
We circumvent our limitations by using *Abstraction* – removal of unimportant details from the picture, irrelevant to the particular consideration.

We apply *Abstraction* in analysis and design of the *Architecture* to the whole system by de-composing it into the finite and manageable number of identifiable components, parts and layers. We continue this process of system de-composition iteratively and recursively, until we reach the well understood basic building blocks of the target *Architecture*. Some iteration in the *Architecture* design may go in the opposite direction, from inside out, when well understood or pre-selected bulding blocks are used to assemble the bigger part of the *Architecture*.

Abstractions may differ by the level of detail. One *Abstraction* can be an expanded drill-down version of another, when finer details come to light. Level of detail in the *Abstraction* determines the *Granularity* of the view. Our models may be *coarse-granular* or *fine granular*.
Levels of Abstraction will complement each other. Higher levels of abstraction will represent the coarse-granular view of the *Architecture*.

Architecture Defined – 'pick one'

Let's take stock of some *Architecture* definitions, including our main focus of interest – *Software Architecture* and *Enterprise Architecture*.

Many of the *Architecture's* 'classic definitions' have been located thanks to the comprehensive collection of quotations in [WWW SEI], [WWW EWITA], and [WWW Bredemeyer].
Check these websites out for many more, both classic or textbook definitions, and definitions from battle-hardened grass-roots practitioners. Good thing about standard definitions – there are so many of them to chose from…

Software Architecture definition in [Bass 1998] and [WWW SEI]:

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.
By 'externally visible' properties, we are referring to those assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. The intent of this definition is that Software Architecture must abstract away some information from the system (otherwise there is no point looking at the architecture, we are simply viewing the entire system) and yet provide enough information to be a basis for analysis, decision making, and hence risk reduction."
"Let's look at some of the implications of this definition in more detail.
First, *architecture defines components*. The architecture embodies information about how the components interact with each other. This means that architecture specifically *omits* content information about components that does not pertain to their interaction.
Second, the definition makes clear that *systems can comprise more than one structure*, and that no one structure holds the irrefutable claim to being ***the*** architecture. By intention, the definition does not specify what architectural components and relationships are. Is a software component an object? A process? A library? A database? A commercial product? It can be any of these things and more.
Third, the definition implies that *every software system has architecture*, because every system can be shown to be composed of components and relations among them.
Fourth, *the behavior of each component is part of the architecture*, insofar as that behavior can be observed or discerned from the point of view of another component. This behavior is what allows components to interact with each other, which is clearly part of the architecture. Hence, most of the box-and-line drawings that are passed off as architectures are in fact not architectures at all. They are simply box-and-line drawings."

[Booch 1999]:
"An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization - these elements and their interfaces, their collaborations, and their composition."

[Jazayeri 2000]:
"Software Architecture is a set of concepts and design decisions about the structure and texture of software that must be made prior to concurrent engineering to enable effective satisfaction of architecturally significant explicit functional and quality requirements and implicit requirements of the product family, the problem, and the solution domains."

From the UML 1.3 Specification [WWW OMG]:
"Architecture is the organizational structure of the system. Architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include classes, components and subsystems."

David Garlan and Dewayne Perry have adopted the following definition for their guest editorial to the April 1995 IEEE Transactions on Software Engineering devoted to Software Architecture:
"The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time."

Writing in 1994 for the ARPA Domain-Specific Software Architecture (DSSA) program, Hayes-Roth says that *Software Architecture* is "...an abstract system specification consisting primarily of functional components described in terms of their behaviors and interfaces, and component-component interconnections".

Barry Boehm and his students at the USC Center for Software Engineering write in 1995:
"A software system architecture comprises:

- A collection of software and system components, connections, and constraints
- A collection of system stakeholders' need statements

- A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements."

By "system stakeholders' need statements" Boehm's definition of the Software Architecture means requirements, both functional and non-functional.

[Hofmeister 2000]:
Based on structures found to be prevalent and influential in the development environment of industrial projects that they studied at Siemens Corporate Research, software architecture has at least four distinct incarnations.
Within each category, the structures describe the system from a different perspective:

- The conceptual architecture describes the system in terms of its major design elements and the relationships among them
- The module interconnection architecture encompasses two orthogonal structures: functional decomposition and layers
- The execution architecture describes the dynamic structure of a system
- The code architecture describes how the source code, binaries, and libraries are organized in the development environment.

[Shaw 1996]:
In 1995, at the First International Workshop on Architectures for Software Systems, Mary Shaw provided a much-needed clarification of the terminological chaos. Distilling the definitions and viewpoints (implicit or explicit) of the workshop's position papers, Shaw classifies the views of software architecture thus:

- *Structural models* all hold that software architecture is composed of components, connections among those components, plus (usually) some other aspect or aspects, including (grouping suggested by the authors):
- Configuration, style
- Constraints, semantics
- Analyses, properties
- Rationale, requirements, stakeholders' needs

Work in this area is exemplified by the development of architectural description languages (ADLs), which are formal languages that facilitate the description of architecture's components and connections. The languages are usually graphical, and provide some form of "box and line" syntax for specifying components and hooking them together.

- *Framework models* are similar to the structural view, but their primary emphasis is on the (usually singular) coherent structure of the whole system, as opposed to concentrating on its composition. Framework models often target specific domains or problem classes. Work that exemplifies the framework view includes domain-specific software architectures, CORBA or CORBA-based architecture models, and domain-specific component repositories.
- *Dynamic models* emphasize the behavioral quality of systems. "Dynamic" may refer to changes in the overall system configuration, setting up or disabling pre-enabled communication or interaction pathways, or the dynamics involved in the progress of the computation, such as changing data values.
- *Process models* focus on construction of the architecture, and the steps or process involved in that construction. In this view, architecture is the result of following a process script. This view is exemplified by work in process programming for deriving architectures.

These views do not preclude each other, nor do they really represent a fundamental conflict about what software architecture is. Instead, they represent a spectrum in the software architecture research community about the emphasis that should be placed on architecture - its constituent parts, the whole entity, the way it behaves once built, or the building of it. Taken together, they form a consensus view of software architecture.

In what has come to be regarded as seminal paper on *Software Architecture*, David Garlan and Mary Shaw suggest in 1993 that Software Architecture is a high level of design concerned with issues beyond simple programming.
"As the size and complexity of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organization and global

control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; composition of design elements; scaling and performance; and selection among design alternatives. This is the software architecture level of design."

Crispen in 1994:
" An Architecture, as we intend to use the term, consists of (a) a partitioning strategy and (b) a coordination strategy. The partitioning strategy leads to dividing the entire system in discrete, non-overlapping parts or components. The coordination strategy leads to explicitly defined interfaces between those parts."

Moriconi in 1994:
"A software architecture is represented using the following concepts:

1. Component: An object with independent existence, e.g., a module, process, procedure, or variable.
2. Interface: A typed object that is a logical point of interaction between a component and its environment.
3. Connector: A typed object relating interface points, components, or both.
4. Configuration: A collection of constraints that wire objects into a specific architecture.
5. Mapping: A relation between the vocabularies and the formulas of an abstract and a concrete architecture. The formula mapping is required because the two architectures can be written in different styles.
6. Architectural style: A style consists of a vocabulary of design elements, a set of well-formedness constraints that mush be satisfied by any architecture written in the style, and a semantic interpretation of the connectors.

Components, interfaces, and connectors are treated as first-class objects - i.e., they have a name and they are refinable. Abstract architectural objects can be decomposed, aggregated, or eliminated in a concrete architecture. The semantics of components is not considered part of architecture, but the semantics of connectors is."

Garlan in 1994:
"A critical aspect of the design for any large software system is its gross structure that is, its high-level organization of computational elements and interactions between those elements. Broadly speaking, we refer to this as the software architectural level of design."

Kruchten in 1994:
"Software architecture deals with the design and implementation of the high-level structure of the software. It is the result of assembling a certain number of architectural elements in some well-chosen forms to satisfy the major functionality and performance requirements such as scalability and availability. Software architecture deals with abstraction, with decomposition and composition, with style and esthetics."

Dewayne Perry and Alexander Wolf in "Foundations for the Study of Software Architecture", ACM SIGSOFT Software Engineering Notes, 17:4, October 1992:
"… Software architecture is a set of architectural (or, if you will, design) elements that have a particular form.
We distinguish three different classes of architectural element:

- Processing elements;
- Data elements; and
- Connecting elements."

[IEEE STD 610.12-1990] on *Architecture* and *System*:
Architecture is defined as "the structure of components, their relationships, and the principles and guidelines governing their design and evolution over time".
"Architecture is the organisational structure of a system."
"A system is a collection of components organised to accomplish a specific function or set of functions."

[Rechtin 1991] on *System Architecture*:
"The term *Architecture* (*System Architecture*) is widely understood and used for what it is – a top-down description of the structure of the *System*."

UML 1.3 definition of a *System*:
"A system is a collection of connected units that are organized to accomplish a specific purpose. A system can be described by one or more models, possibly from different viewpoints."

[ANSI/IEEE Std 1471-2000], Standard for Architectural Description of Software-Intensive Systems on *System Architecture*:
"The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution."

Ruth Malan [WWW Bredemeyer] on the *Enterprise Architecture*:
"An enterprise is made up of many interacting systems of various kinds. *Enterprise Architecture* identifies these systems, their key properties, and their interrelationships, and plans for and guides the evolution of the enterprise systems to support and enable the evolution of the enterprise in its pursuit of strategic advantage.
Thus, *Enterprise Architecture* is fundamentally is a 'system of systems' architecture, while *Software Architecture* is a 'system of components' architecture (where systems produce stand-alone value, and components have to be composed into a system to produce value)."

The Open Group's definition of the *Enterprise Architecture* in [WWW TOGAF]:
"There are four types of architecture that are commonly accepted as subsets of an overall Enterprise Architecture:

- Business Architecture: this defines the business strategy, governance, organization, and key business processes.
- Data/Information Architecture: this describes the structure of an organisation's logical and physical data assets and data management resources.
- Application (Systems) Architecture: this kind of architecture provides a blueprint for the individual application systems to be deployed, their interactions, and their relationships to the core business processes of the organization.
- Information Technology (IT) Architecture: the software infrastructure intended to support the deployment of core, mission-critical applications. This type of software is sometimes referred to as 'middleware', and the architecture as a 'technical architecture'."

Architecture Reference Models

This section outlines some of the more representative complete conceptual frameworks for analysis and design of Software and Enterprise Architectures.

Reference Model of Open Distributed Processing (RM-ODP)

The Reference Model of Open Distributed Processing (RM-ODP) describes common interaction model for heterogeneous distributed systems ensuring their interworking, interoperability, and portability. The RM-ODP standard is known as both ISO International Standard 10746 and ITU-T (formerly CCITT) X.900 Series of Recommendations.

Stated goals of the RM-ODP are:

- Portability of applications across heterogeneous platforms
- Interworking between distributed components of the system by meaningful exchange of information and convenient use of functionality throughout the distributed system
- Distribution transparency by hiding the distribution and remoteness of components from the application programmers and users

RM-ODP defines five abstract *viewpoints* for describing the distributed systems:

- ***Enterprise Viewpoint*** – purpose, scope, business objects and policies
- ***Information Viewpoint*** – semantics of information processing, and data model schemas of three kinds – static, invariant, and dynamic schemas

- ***Computational Viewpoint*** – object-based functional decomposition of the distributed system, interfaces and behaviours
- ***Engineering Viewpoint*** – infrastructure required to support distribution using the notation of *objects* and *channels*. Channel provides communication mechanisms and controls transparency functions
- ***Technology Viewpoint*** – choices of technology for implementation

RM-ODP defines set of *functions* or common services for management, transactions, persistence, objects registry and location, security.

RM-ODP defines an initial extensible set of *transparencies* that hide the distribution and heterogeneity of the system from the user:

- ***Access Transparency*** – hides the differences in data representation and access procedures
- ***Location Transparency*** – hides distinction between local versus remote
- ***Relocation Transparency*** – hides the relocation of the object transparent to its clients
- ***Migration Transparency*** – masks the relocation of the object from that object and its clients
- ***Persistence Transparency*** – masks the activation and deactivation of an object from the storage
- ***Failure Transparency*** – masks the failure and recovery of the objects for fault tolerance
- ***Replication Transparency*** – hides group of replica objects from the common interface
- ***Transaction Transparency*** – hides the coordination required to satisfy the transactional integrity of operations

Catalysis

Catalysis is the standards-based methodology for the systematic development of object and components-based systems.
Original developers of the Catalysis methodology are Desmond D'Souza and Alan Wills [D'Souza 1999].

Catalysis method has a simple core, covering three *Levels of Description*, using three basic *Modelling Constructs*, applying three underlying *Principles*.

Three *Levels of Description*, or *Modelling Scopes*:

- ***Domain/Business Models.*** For any component, a domain model describes the context in which that component will reside. Domain model identifies the problem and establishes the problem domain terminology. Domain model helps understand business processes, roles and collaborations. Methodology can build distinct *as-is*, or models describing existing business context, and *to-be* models. Domain models can be of interest independent of any software system or component
- ***Component External Specifications.*** Component specification describes externally visible behaviour of component with its environment, and defines component responsibilities and interfaces
- ***Component Internal Design.*** Internal design model defines internal architecture of components and how the external behaviour is actually implemented. Architect will exercise own judgement on how detailed a dissection of the component, and how deep a drill-down into the guts of internal design is required

Catalysis method uses three basic *Modelling Constructs*. Or, rather, two primary modelling constructs (*Collaboration* and *Type*) complemented by construct for levels of abstraction (*Refinement*) and composition patterns for all other constructs (*Framework*):

- ***Collaboration.*** Specifies behaviour of group of objects
- ***Type.*** Specifies externally visible behaviour of an object, abstracting from details of its internal design, algorithms and data structures
- ***Refinement.*** Relates different levels of detail or abstraction in the description of behaviour. Provides mapping and traceability for each element between different levels
- ***Framework.*** Captures generic recurring patterns of *Collaborations*, *Types* and *Refinements*.

Three *Principles* of Catalysis are:

- ***Abstraction.*** Focuses on essential aspects, aims for uncluttered description of requirements and architecture. Encourages developer to stay on higher level of abstraction whenever possible. Dealing with complexities of the software system on the code level may be overwhelming. Examples of common abstractions are interfaces (which hide implementations details), and architectural patterns (which hide details of usage in different domains)
- ***Precision.*** Exposes gaps and inconsistencies early, makes abstract models accurate
- ***Pluggable Parts.*** Most of the work done by adapting and composing existing or off-the-shelf parts or components. Models, architectures and designs are assets. Catalysis aims to "reduce magic" and nurtures re-use of components

Constructs and *Principles* recursively apply at all *Levels*.

Catalysis gives its definition of *Architecture* as the set of principles for design decisions, rules, or patterns about any system that keeps its designers from exercising *needless creativity*.
Catalysis aims to *minimize the magic* that happens in the software development process.

The Open Group Architectural Framework (TOGAF)

In 1984, group of European vendors formed X/Open consortium committed to promoting the independent standards and achieving portability of IT solutions. X/Open gained worldwide recognition and published series of Portability Guides (XPGn).

Open Software Foundation (OSF) was founded in 1988 by major system vendors in support for not-for-profit international research and development.

The Open Group consortium came into life by amalgamation of efforts of two consortiums – X/Open and OSF. [WWW OpenGroup]

The Open Group aims to bring together customers and suppliers of information systems while keeping neutrality towards vendors, products and technologies.
The Open Group develops and deploys frameworks, policies, best practices, standards, and conformance programs that address needs of customers and, at the same time, reconcile and guide efforts of many vendors of information technology.

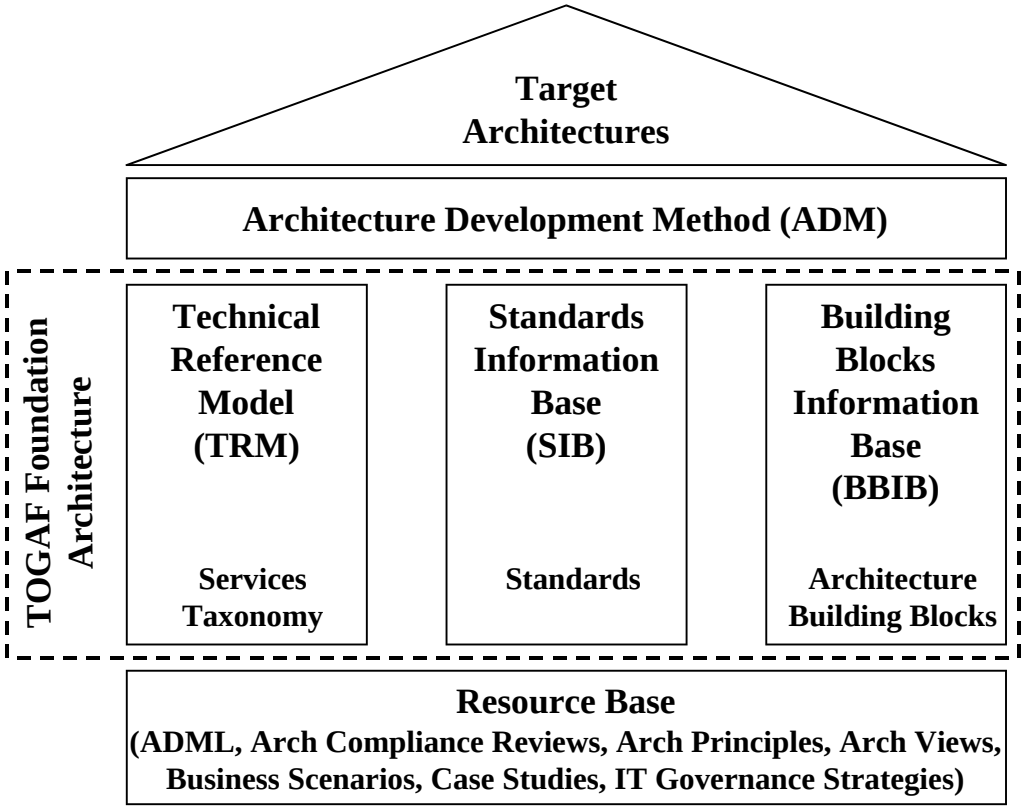
The Open Group pursues the vision of making all technology open, accessible, and interoperable. "Anyone will be able to access anything to which they are entitled from anywhere at anytime".

The Open Group Architecture Framework (TOGAF) started its history from 1994 with definition of requirements. The Open Group delivered new version of TOGAF every year ever since. In 2001, TOGAF Version 7 was released.
To the large extent, TOGAF leverages IT systems engineering advances in defense and aerospace industries that historically were at the forefront of architectural thinking. However, major commercial vendors and stakeholders increase their share of contributions.

TOGAF is an architectural framework consisting of two parts:

- ***The TOGAF Architecture Development Method (ADM)***, which explains how to derive organization-specific IT architecture that addresses business requirements. "The key to TOGAF is the TOGAF Architecture Development Method (ADM) – a reliable, proven method for developing an IT Architecture that meets the needs of your business" [WWW TOGAF, Frequently Asked Questions]. ADM provides architecture views and tools for architecture development. The Open Group embarked on development of standard for Architecture Description Markup Language (ADML) – XML-based language for describing software architectures to enable their representation, evaluation and analysis.
- ***The TOGAF Foundation Architecture*** that includes The TOGAF Standards Information Base (SIB), a database of open industry standards for services and components. Building Blocks Information Base (BBIB) - re-useable architectural building blocks. Foundation Architecture defines TOGAF Technical Reference Model (TRM) that introduces taxonomy and scope of ***services*** and system-wide capabilities, or ***qualities*** (like Security, Transaction Processing, or

Data Management). TOGAF provides Architecture Principles, Architecture Views and Business Scenarios as architect's tools and resources.



Source: TOGAF

Figure 5.1. TOGAF Structure and Components

TOGAF defines overall Enterprise IT Architecture as consisting of four closely interrelated Architectures – Business Architecture, Data/Information Architecture, Application Architecture, and Technology (IT) Architecture.

TOGAF recognizes the fact that there is no single universal architecture suitable for all purposes at all times. There is a continuum of architectures, and IT architects require design tools in the form of architectural framework.
Architectural Framework embodies best practices and acknowledged wisdom, and guides the development of specific architectures. Framework does not make architectural design an automatic process, but provides a valuable aid to experienced IT Architects.

Zachman Institute for Framework Advancement (ZIFA)

Zachman Framework [WWW ZIFA] is a high-level framework that describes architecture elements.

Zachman Framework can be used to describe any other framework by mapping to the defined architecture elements. Zachman Framework has consistently proven itself in building many large-scale Enterprise Architectures, and in use by architecture tool vendors (eg. Ptech, Popkin).

In words of John Zachman: “The Zachman Framework is a total set of descriptive representations to fully describe a complex object”.

Zachman Framework can be visualised as a two-dimensional model that defines various *Abstractions* (or aspects, columns in the matrix) and *Perspectives* (or viewpoints on a given aspect, rows of the matrix). Cell in the matrix on the intersection of particular *Abstraction* and *Perspective* defines a specific model for documenting the architecture design.
Framework guides architects towards verifiable completeness of vision of the Enterprise Architecture, and determining required work products for documenting the architecture.

Abstractions / Perspectives	DATA (What)	FUNCTION, PROCESS (How)	Location, NETWORK (Where)	PEOPLE, Organization (Who)	TIME, SCHEDULE (When)	Motivation, STRATEGY (Why)
SCOPE (Planner’s View, Contextual)	List of Things Important to Business Entity=Class of Business Thing	List of Processes the Business Performs Function=Class of Business Process	List of Locations Important to Business Node=Major Business Location	List of Organizations Important to Business Agent=Major Org Unit	List of Events Significant to Business Time=Major Business Event	List of Business Goals/Strategies End/Means=Major Bus Goal / Critical Success Factors
ENTERPRISE MODEL (Owner’s View, Conceptual)	e.g., Entity-Rel. Diagram, Semantic Model Ent=Business Entity Rel=Business Rel.	e.g., Function Flow Diagram, Bus. Process Model Function=Bus. Process I/O=Bus. Resources	e.g., Logistics Network Node=Business Location Link=Business Linkage	e.g., Organization Chart Agent=Org Unit Work=Work Product	e.g., Master Schedule Time= Business Event Cycle=Business Cycle	e.g., Business Plan End=Bus Objectives Means=Bus Strategy
SYSTEM MODEL (Designer’s View, Logical)	e.g., Logical Data Model Entity=Data Entity Relationship= Data Relationship	e.g., Data Flow Diagram, App. Arch Funct=Appl Function I/O=User Views	e.g., Distributed System Architecture Node=Info Sys Funct Link=Line Char	e.g., Human Interface Architecture Agent=Role Work=Deliverable	e.g., Processing Structure Time=System Event Cycle=Processing Cycle	e.g., Knowledge Arch, Bus. Rule Model End=Structure Assert Means=Action Assert
TECHNOLOGY CONSTRAINED MODEL (Builder’s View, Physical)	e.g., Physical Data Model Entity=Table/Segment Rel=Key/Pointer	e.g., Structure Chart, System Design Funct=Computer Funct I/O=Screen/Device Formats	e.g., System Arch, Technical Arch Node=Hardware/ System Software Link=Line Spec	e.g., Human/ Technology Interface Agent=User Work=Job	e.g., Control Structure Time=Execute Cycle=Component Cycle	e.g., Knowledge Design, Rule Design End=Condition Means=Action
DETAILED REPRESENTATION (Sub-Contractor’s View, Out-Of-Context)	e.g., Data Definition Ent=Fields Rel=Addresses	e.g., Program Funct=Language Stmts I/O=Control Blocks	e.g., Network Architecture Node=Addresses Link=Protocols	e.g., Security Architecture Agent=Identity Work=Transaction	e.g., Timing Definition Time=Interrupt Cycle=Machine Cycle	e.g., Knowledge Definition, Rule Spec End=Subcondition Means=Step
FUNCTIONING ENTERPRISE						

© 2003 SAFE House

Figure 5.2. Concepts of the Zachman Framework for Enterprise Architecture

That’s not all, folks…

Having observed many views and definitions of the *Architecture*, we could not resist in the heading of this section above but paraphrase the saying of our favourite rabbit…

Facing variety of specific challenges in building the *Enterprise Architecture*, *Architects* all over the world absorb some canonical definitions and methodologies, and embark on the task of adjusting and adopting these visions better to the problem at hand, and rightfully so.

Whole industries (like public services, military, or health) and large Enterprises may have architecture concepts and methodologies well standardised, making life of the Software Architect somewhat easier

by reducing upfront the multitude of hard choices that Architect has to make, and by aligning all major players in the Enterprise Architecture behind the one consistent vision.
One may criticize the particular framework or methodology. Framework may work better or worse in every particular situation, but the introduction into the large and complex project, with many stakeholders and agendas, of the cohesive big-picture vision, is a good thing – if nothing else, good for narrowing down the problem area.

Architectural Framework is the important pre-requisite for successful risk management in building the Enterprise Architecture.
It may take time, efforts, and persistence to have an architectural framework for your industry, enterprise, or application domain nailed down and agreed upon by main stakeholders. But, as soon as framework is successfully utilized at least once with all infrastructure put in place and experiences accumulated, it can only get easier, better, and cheaper second and third time around (Product Line for the Enterprise Architecture if you like).

Software and Enterprise Architecture is a fluid field, constantly evolving with advances in the Information Technology. In addition, Software Architecture challenges may vary depending on its place in the core business of the Enterprise, and the level of buy-in.
Many, even sizeable, enterprises will not impose on the staff Architect or consultant the prescribed framework for building the Enterprise Architecture. In no way this means that Software and Enterprise Architect can release him/herself from the responsibility of devising the systematic vision for the Enterprise Architecture that strives to achieve its business goals. Best contribution of the Software Architect to the Enterprise may well be the definition of such a vision.
No Enterprise can afford a free-fall approach in strategic technological decisions – risk is just too high for the Enterprise that has the critical dependency of its core business and its competitive advantage on the Information Technology.
Software or Enterprise Architects' mistakes are much more damaging and costly than the ones of a Programmer.

We take our hats off before scores of practicing Software Architects and Enterprise Architects out there in the field, who often face a challenge of not just delivering a single project, but putting a house in order by delivering an architecture vision for their Enterprise, and by advancing and enriching the collective architecture knowledge base in the process.

Local efforts may range from the meticulous application of the known methodology, to its significant revision and adaptation to the specific project requirements and team experiences, through to devising own vision and methodology based on industry experiences. Latter scenario is possible, but generally better to be avoided – rarely your project will be a 'green fields' to such an extent.
Nevertheless, limited (possibly, temporarily) adoption of some 'home grown' architecture frameworks does not diminish their value as contributors to our Architect's Toolbox.

As an example of yet another complete systematic vision for the Enterprise Architecture framework and Reference Model, refer to [WWW Bredemeyer] by practicing architects and educators Dana Bredemeyer and Ruth Malan. Website generously provides original articles with architecture resources for download.
Rest of the section is broadly based on the main architectural points from this source.
<<< Request permission for digest, possibly drawings >>>

Enterprise Architecture

Enterprise Architecture:

- Encompasses and relates constituent architectures:
  - Application/Software Architecture
  - Technical/IT Architecture (the common platform or shared infrastructure, components, frameworks, tools)
  - Information/Data Architecture
  - Organization/Business Architecture
- Provides the vision and guiding principles that govern all of these architectures
- Addresses enterprise-level objectives like:

- Consistency and cohesion, integration, interoperability, security
- Flexibility to make, buy or outsource IT solutions
- Re-use across applications, product lines or product families

As we see, *Enterprise Architecture* defined in terms more general architecture concepts and views – *System Architecture*, *Software Architecture*, *Technical Architecture*. This way we introduce high-level taxonomy and vocabulary for the *Enterprise Architecture*.

System Architecture

A *System* is defined as a set of different elements so connected or related as to perform a unique function not performable by elements alone. This implies that the whole is greater than the sum of the parts, that is, the *System* has properties beyond those parts.

Software Architecture

We define our architectural models as a set of complementary interrelated views (possibly in different orthogonal dimensions) and layers.

Software Architecture views:

- ***Conceptual Architecture.*** Focuses on identifying components and allocating responsibilities to components. Provides decomposition of the system without delving into details. Consists of the Architecture Diagram (without interfaces) and informal component specification for each component. Provides useful vehicle for communicating the architecture to non-technical audiences, such as management, marketing, and users
- ***Logical Architecture.*** Focuses on component interactions, connection mechanisms and protocols. Adds precision to the Architecture Diagram by detailing interfaces and component collaboration. Establishes contracts by which separate components and development streams may progress relatively independent
- ***Execution Architecture.*** Assigns runtime component instances to processes, threads and address spaces. Allocates physical resources. Shows process view and deployment view of the physical system

And again, *Software Architecture* views along another dimension:

- ***Structural View.*** Consists of Architecture Diagram, Component and Interface Specifications. Captures static view of architecture
- ***Behavioural View.*** Consists of Component Collaboration or Sequence Diagrams. Captures timeliness and dynamic view of the system

Every view of the *Software Architecture* may be analysed to the certain depth of detail, corresponding to the *Software Architecture Layers*:

- ***Meta-Architecture.*** Architectural vision, principles, styles, key concepts and guidelines. Provides high-level constrains that strongly influence structure of the system and rules certain structural choices out. Guides selection decisions and trade-offs
- ***Architecture.*** Components and relationships, static and dynamic views. Conceptual, Logical and Execution Architectures, as described above
- ***Architecture Guidelines and Policies.*** Design patterns and policies, frameworks, infrastructure and standards. Guides designers and engineers in creating designs that maintain the integrity of the architecture

Technical Architecture

[WWW OpenGroup] defines *Technical* or *IT Architecture* in The Open Group Architecture Framework (TOGAF) as formal description of the Information Technology system, organised in a way that supports reasoning about the structural properties of the system.
IT Architecture defines the components or building blocks that make up an overall information system within the technology constraints, and provides a plan from which products can be procured, and systems developed, that will work together to implement the overall system.

Software Architect's Profession

Who Is the Software Architect?

One cannot learn acrobatics solely by reading the manual. In order to achieve reasonable proficiency in acrobatics, you have to train hard, and to practice it, constantly. There are good acrobats, and there are bad ones. You may not be a good acrobat yourself, but you surely can recognise one when you see him or her. And, someone just is not cut out to become an acrobat at all.
This is true for any type of human activity that requires high level of schooling, commitment, attitude, expertise, and talent – builder, sports person, programmer, teacher etc. Software Architecture is not an exception in this respect.
Although, Software Architecture activities exhibit less immediately visible qualities than, say, acrobatics. Possibly, it would be easier to pass for the Software Architect for some time, while not being one.

We heard about impostors posing as brain surgeons. One would not go under their scalpel knowingly. Given time and opportunity, they may well become very good brain surgeons indeed, but they are not at the moment.
We would look for the proven track record of previous successes and credibility, before we entrust our health to the medical practitioner. Why Software Architecture should be treated any different when we entrust to it the livelihood of the whole core business of the Enterprise?

IT industry offers plethora of tags for the practitioners with broad expertise – Software Architect, Software Engineer, Systems Architect, Systems Engineer, IT Solutions Architect, Application Architect, Domain Architect etc. etc.
These notions are strongly related and overlap significantly, and constantly move in or out of fashion in some pockets of industry.

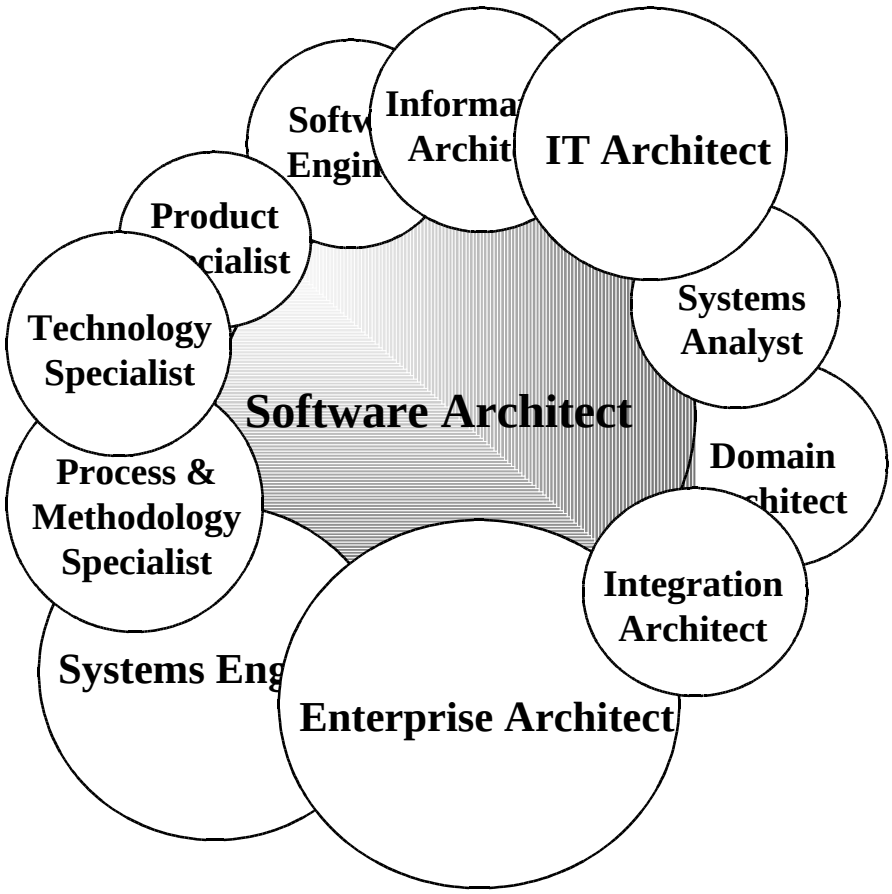


Figure 5.3. Overlapping Roles in Software Architecture

Probably, most generic and, at the same time, most abused term with IT connotations is *System*.
We can (and will) slice and dice the notion of *System* to components and layers, entities and relationships, interfaces, inputs and outputs etc., but the most general and enduring definition of *System* is this – *System* is anything that is worth considering. And, by any*thing*, we do not necessarily mean the material *thing* here.
Consequently, most generic terms to describe the IT professional, who is involved in various aspects of *Systems Engineering* using IT, are *Systems Engineer* or *Systems Architect*.

Software Architecture may be viewed as part of the overarching *Systems Engineering* discipline. However, *Software Architecture* looks closely into context in which actual *Software* runs – hardware, networks, protocols, interfaces and other pieces of the puzzle that may be beyond the immediate interest of the software programmer. This broad view on the *Software Architecture* makes it almost indistinguishable in the IT-intensive field from the whole discipline of *Systems Engineering*.
And, you guessed this right - *Software Architects* are people who do *Software Architecture*.

There is long way from the bare computer chip and binary 0s and 1s, to the coarse-granular components that perform meaningful function for the humans in the real world – from the IT models on the *micro*-level of abstraction (in internal interworkings of our computer systems), to the *macro*-level business components and frameworks (where we engage in actual interaction with the real world we build computer systems for, i.e. where rubber hits the road, if you like).

We may have lost on a *macro*-level the precision and order that we praise ourselves for on the *micro*-level.

Business' perception may well be that we have lost the plot… It is high time to revert that perception.

Why Good Architecture is Important?

Good Architecture is the one that scores high on the matrix of Quality Measures defined earlier in the book. We do not define in this section what makes architecture good. But we do want to focus on why the good architecture is important.

Good Architecture enhances productivity. Good separation of concerns in architecture components and interfaces enables parallel development, in separate streams of work, by separate teams, vendors or outside contractors

Good Architecture contains costs. Re-use of expensive components, infrastructure, skills and processes, reduced overall complexity – all directly translate into reduction of costs, to the point of getting into the green zone of solution being financially feasible in the first place.

Good Architecture reduces risks. By making the successful delivery on time and within budget predictable with high level of comfort, we remain in the control of the process and minimise probability of unpleasant surprises at the time when business committed already, exposed to dangers of failure and may not have enough available options to react. Also, success in partial solution does not open the gaping hole somewhere on the bigger picture in your business context.

Good Architecture improves flexibility, resilience, adaptability. Changing technologies, changing business requirements, updates and enhancements may quickly cause the 'architecture rot' when architecture turns bad. Goodness of the Architecture must have its own lifespan during which business should not feel the stench of the rotten architecture, until architecture revamped and improved in a planned and orderly fashion.

Good Architecture improves manageability by easing maintenance, support and enhancements in the overall system throughout its life cycle.

Good Architecture improves agility in responding to changing business needs in timely fashion and within budget.

Good Architecture improves professional satisfaction and fosters better working environment and teamwork.

Many costly and embarrassing failures, horror stories and disasters owe the great deal to the Bad Architecture that was the manifestation, and the cause of the failure.

What Makes an Architect Good?

We understand by now that making the good *Enterprise Architecture* is hard, and requires knowledge and skills in many disciplines.
So, ideally, who are these Super Humans we call Enterprise Architects? And they really are humans – there will be no alternative here for the foreseeable future, fortunately (as there won't be an ideal single person – perfect Enterprise Architect either, only well-rounded at best, as we shall see).

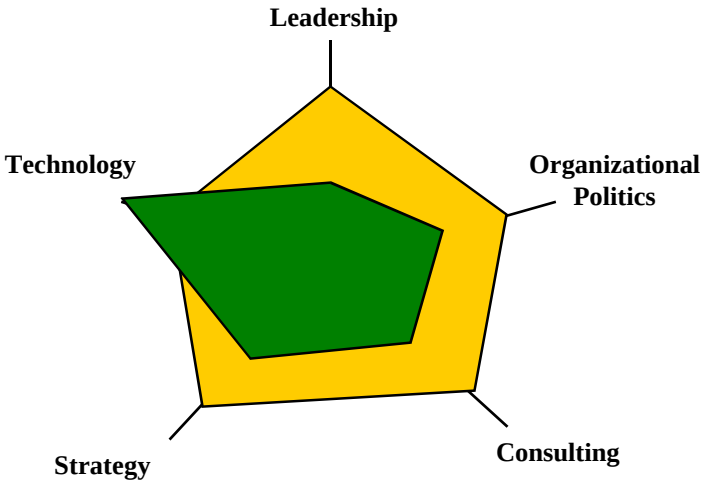
Architect Competency Model

Dana Bredemeyer [WWW Bredemeyer] provided very insightful and convincing arguments by presenting the Architect Competency Model, Action Guides for the Enterprise Architect, and the refreshing overall architect's approach called L/F/G Frame of Reference.
('L/F/G' stands for Lead, Follow, Get Out of the Way – all of which necessary to realise the architecture vision, to build the capable and focused team, and, sometimes, is just the right thing to do).
This section is largely based on Dana's presentations.
<<< Request permission >>>

Dana Bredemeyer defines five primary roles played by the architect. For each role Dana defines “What you KNOW”, “What you DO” and “What you ARE” as an architect.
Five roles, or “5 Hats of the Architect” are:

- ***Technology.*** You need a thorough knowledge of relevant technologies, your business’s product domain and processes. The problems may not be well defined, unlike ones that developers facing, often with unclear or conflicting objectives. The personal characteristics really essential to success in this domain are a high tolerance for ambiguity and a lot of skill working consistently at an abstract level
- ***Strategy.*** Solid understanding of your organization’s business strategy and rationale behind it is the key. As a skilled technologist you create good architecture. As a skilled strategist, you create the right architecture for your organization
- ***Leadership.*** Architecture team without leadership goes nowhere. A leader is required to infuse the team with a common vision, and to motivate the core team and associated teams to do their best work. This requires dedication and passion, a strong belief that you can lead the effort and the desire to do so. You must see yourself, and other must see you, as a credible leader
- ***Consulting.*** The actual users of architecture are development teams. While using architecture may be the best overall approach for the organization, this is often not apparent to its users. You are functioning here as a mentor and teacher. What really contributes to your success here is to be truly committed to others’ success and to have a good understanding of change management and how groups adopt new processes
- ***Organisational Politics.*** Architectures have many and diverse stakeholders. You really need to understand business and personal objectives of key players, and get them personally committed to the success of the architecture. This means listening, networking, articulating and selling the vision, and doing all this continuously over the life of the project. The people doing this well are extremely articulate and confident. They are resilient and driven, and they are sensitive to where the real power is and how it flows. They look for and see the organization behind the organization, and they use this insight to build and maintain support for their projects

Figure 5.3 shows Kiviat, or spider diagram with roles of the Enterprise Architect, and how he or she fares. Diagram was adapted from Dana Bredemeyer’s presentations, and originally focused on the ‘lead architect’ competency. This competency model is applicable for the Enterprise Architect as well. Enterprise Architect has to score high on every role. You may use this diagram for self-assessment to see if you are well suited for the job.



Legend:

Poor candidate for Enterprise Architect

Good candidate for Enterprise Architect

Source: Bredemeyer

Figure 5.3. Enterprise Architect Competency

Taxonomy of Architects

Let's take a somewhat light-hearted look at the vibrant community of Enterprise Architects, and identify common archetypes that exist there.

Each Architect is a person of many talents, and quite capable of defying any simplistic labelling. Reliable certification of the Enterprise Architect is on the wish list in IT industry. The more important then is the ability of the Architect for the realistic, if not humorous, self-assessment.

Likely, the Architect, that you might come across in your project, falls into one or more of the following categories or types (features are intentionally exaggerated):

- ***'Loose Cannon'***. Has some expertise in some relevant areas. Knows most of the right words, learns them quickly, but often uses them in wrong context or order, without any qualms. Defensive and aggressive. Prepared to make hasty shortcuts. Has low level of tolerance to perceived failings of others.
- ***'Guru'***. Knowledgeable and experienced. Proud authority, Subject Matter Expert in technology, application domain, or in the line of business. Afraid of making mistakes. Often unable to promote and defend the right decision. If different solution was selected against his preference (possibly, due to some overriding commercial considerations), expect to hear from him/her 'I told you so' in case of any usual hiccup. There is always somebody else's fault.

- ***'Rocket Scientist'***. Perfectionist and the religious follower of the fundamental theory and scientific rigour. See 'Guru', only with problems in keeping feet on the ground, in proposing viable and feasible solutions, with acceptance of necessary practical compromises.
- ***'Talking Head'***. Adept in the organisational politics. Usually harmless, if not helpful. Can assist in the troubled project greatly by calming down the situation, and by bringing the good old street-wise common sense.
- ***'Prince'***. Prefers to 'facilitate' and 'review', rather than perform actual solution design and implementation hands-on. Often out of the depth professionally, but maintains the smokescreen. Can give a hard time to the external vendors of Professional Services, as their options in managing the situation are limited.
- ***'Czar'***, or ***'Warlord'***. Arrogant and intolerant. See 'Prince', but more aggressive and malignant. Demonstrates vastly different behaviour in managing the situation 'up' and 'down' the organisational hierarchy, as well as sideways to various stakeholders. Will not share information voluntarily. Capable of passing someone else's intellectual property for his own.
- ***'Seagull'***. Usually, this type possesses superficial expertise, combined with high self-appreciation and admirable survivability. Prefers short-term forays into project, before tangible deliverables take shape. Covers himself (or herself) with voluminous paperwork, and by properly following official guidelines, often without much substance. Makes a lot of noise and leaves droppings everywhere. Quickly takes off to the new pastures before things get real ugly.
- ***'Working Horse'***, or ***'Rock'***. Knowledgeable, result-oriented, and helpful all-rounder and team player. Consistent contributor to the team deliverables. Capable promoter of adopted architectural decisions to management, peers, and any other stakeholder. Does not buckle under pressure, responds to it by increasing own performance (cheerfully, clenching the teeth, or both) and making himself available to the team for help. Looks out for organisational politics, but does not make it a main pre-occupation. Every project manager must have one.
- ***'Pillar'***. See 'Rock', with greater leadership strengths. Top-notch professional, with highest level of work ethics. Accumulated broad technical expertise through vast and diverse industry experience in various technologies, methodologies, and roles. Technical expertise can get rusty in places, but solid understanding of the technology principles is still there. Also possesses the penetrating power and the leadership ability. Diffuses project pressures and tensions before smoke turns to fire. Gets work done, and without making enemies along the way. Privilege and pleasure to work with.

One might comment that most of these categories sound pretty negative. This does not mean that the most of us Architects are bad, or belong to one category. We did not mean for you to take this *Taxonomy of Architects* too seriously. On the other hand… Every good joke rings some truth. It is much better for all concerned, if truth surfaces from the well-meaning benign joke.
You are half way there already if you know where you stand. If you recognise yourself in any of these categories, not everything is lost.

If we step further back, we might ask ourselves the question: "What exactly is 'work' for a person who is paid to think?" - as scribbled in the margins of the book by one of the industry luminaries, the creator of Dilbert, who helps to bring the breath of fresh air in what we all do [Adams 1997].
It always pays to see a bigger picture, or forest for the trees, or to keep feet on the ground, or….
Enough already, you've got the idea.

Revolution and Evolution

Advances in Information Technology, and its wide spread adoption represent one of the most amazing revolutions in technology and society in the last century.
However, qualitative leap in technology and society at large happened through gradual accumulation of evolutionary breakthroughs and achievements in some pockets of technology.
What seems, or proclaimed to be a revolution in some highly specialised field, may look like a storm in a teacup from a broader prospective. Or, upon closer inspection, may seem more like marketing ploy with particular agenda.

Software and Enterprise Architecture comprises multitude of IT-intensive technologies and methodologies in their application to the great variety of business and social human activities.

Makeup of business and social processes, and our understanding of them on a *macro*-level, does not change as rapidly as technological advances in the narrower field.
Revolutionary developments in certain technologies will impact, but not necessarily in dramatic revolutionary sense, the landscape of Software Architecture and Enterprise Architecture.

Software and Enterprise Architecture field will never be dull for its practitioners, but one should not expect big surprises on a *macro*-level either.
Software Architecture, as a consolidating multi-discipline framework, provides (as the framework should) a more persistent overarching vision, to some extent immune to specific technological advances.

Software Architects will keenly follow revolutionary advances in technology, methodology, and products, and find the rightful place for these advances in the overall scheme of things.
As soon as this informed and balanced vision is achieved, Software Architect is in position to devise good, fit-for-purpose, feasible, minimum-risk, flexible, resilient to change and 'future-proof' recommendations for the Enterprise Architecture.

<<< … >>>