Table of Contents for CIO Summary in Part 2

Part 2. Architect's Toolbox - 'bird's view of the terrain'

<<< … >>>

CIO Summary

Having discussed key concepts and vocabulary of Software Architecture in Part 1, we embarked in Part 2 on more in-depth overview of methodologies and technologies that summarily represent the palette of Software Architect's tools of trade.

In other words, we peek into Software Architect's Toolbox and learn what these tools are, and how they work.
Of course, as with any powerful tool, it can be useless or even harmful in the hands of barbarian. Furthermore, one should select right tools for the job. You would not use a sledgehammer where you need a scalpel; you would not use a scalpel to cut wires.
Like any proficient craftsperson, Software Architect must know his or her tools well, and use them appropriately.

Every subheading in this part warrants a separate voluminous publication that would explain given methodology or technology in detail it deserves. And, in most cases, you should be able to find such a source of detailed information on the subject (this book will point you to such sources).
This part concentrates on the task of grasping the big picture in one broad swipe, without compromising the rigour of presentation and, at the same time, maintaining the measured level of detail. We aim to explain the concept sufficiently for the understanding of its basics and positioning, and to keep the volume and noise to the minimum (without making the presentation too dry and indigestible, hopefully).
To find such a fine balance in such a broad scope is a very challenging task indeed, but most rewarding. We believe that involved reader will appreciate no-nonsense, high density, and relatively resilient to aging presentation of our evolving field (as author himself, keen reader and hunter for good reference resources, would).

Every self-improving practitioner will have some form of catalogue of hunted down resources, and notes or cheat-sheets with once researched, digested, and understood information on variety of special subjects. Given sheer volume, complexity and dynamicity, this pile will grow on you.
Consider this book a sorted out notes from the desk of such a practitioner, who had to do this research work for himself, and decided to share the outcome with peers facing the same mammoth task. In addition, plain research and summarisation would not have enough practical value if we failed to view every technology through the lens of practical experience and true positioning against other related technologies.

In Chapter 7, we look at major methodologies that gradually summarised and refined our understanding of the metaphors and processes of analysis, design and construction of IT solutions, with some history of the constantly evolving methodologies for re-enforcing the lessons of the past.

Methodologies are forward-looking guidelines, based on past experiences of hits and misses in the construction of Software Architectures.
Methodologies, the digest of previous experiences and accumulated wisdom of hard learned lessons, help us in setting up the frame of reference and the orderly process for achieving the predictably good quality of the solution, with predictable budget and resources, in predictable timeframes.

Software Architecture often aims to deliver a mission-critical component for the core business of the Enterprise. At the same time, building the Software Architecture is sometimes a volatile, intricate, even whimsical activity, likely to be prolonged and resource-hungry as well.
In other words, building Software Architecture is a high-risk activity, and methodologies help us managing the risk better.

We start from the lessons of *Modular and Structured Programming* of 1960s and 1970s. They largely remain current for programmers and software designers despite tremendous advances in programming theory and practices in following decades.
Our growing understanding did not override, but expanded the programming concepts by building on the foundation of Modular and Structured Programming. Basic programming structures have become an intrinsic part of modern programming languages and the common programming practice.

Natural progression of Modular and Structured Programming concepts in 1980s brought us the *Object-Oriented* paradigm, where program constructs have become self-contained entities with their own state and behaviours.
Run-time instance of such program entity is called *Object*. Type or, if you like, the pre-defined shape of similar (in a sense of state and behaviour) objects is called *Class*.

Object-Oriented programming paradigm gave birth to the notation and methodologies of *Object-Oriented Analysis and Design*. Methodologies and notations wars concluded with *Unified Modelling Language* (UML) emerging as a commonly accepted standard for documenting software structures. We describe fundamentals of UML in this book.

UML notation is a tool for describing the Software Architecture. However, it has to be complemented by some methodology to address the needs of defining the process of software construction.
Strictly speaking, software notation and the methodology for software construction are not directly related, but complement each other.
We explain basics of Rational Unified Process (RUP), eXtreme Programming (XP), Agile Modelling, Capability Maturity Model Integration (CMMI), and Patterns and Frameworks in general.

In Chapter 8, we arbitrarily select standards, protocols and related technologies that are of great importance for the Internet-based Enterprise IT Architectures. Again, we were facing the challenge of drawing the line in determining the makeup and scope of selected most important topics, but we had to start somewhere by applying the criteria of greatest pragmatic value for the Software Architect in most practical situations he or she is likely to encounter on real life projects in the Enterprise.

Despite plethora of textbooks on the subject, we could not ignore completely basics of OSI Model and TCP/IP. Feel free to skip this brief section if you are tired to death reading this material elsewhere. Still, we present here to you a convenient digest of necessary fundamentals.

The same goes for HTTP, HTML, cookies, URL, Browsers and Web Servers, and basics of firewalls. You probably take it for granted like the air you breathe. If not, this book will bring you up to speed.

Common Object Request Broker Architecture (CORBA) from Object Management Group (OMG) provides a good example of the message bus architecture for the open distributed object middleware. CORBA solutions may be losing momentum in the Enterprise, but remain strong offering for integration and a good vehicle for understanding concepts and challenges of the distributed computing.
Model Driven Architecture (MDA) from OMG takes the rigour of modelling and system engineering from the message bus in distributed computing one step further – into the realm of platform-independent executable models and inter-operatable components.

Then comes eXtensible Markup Language (XML)…
We explain fundamentals of XML and more important standards from the XML suite that will help you to grasp the really simple concept of XML, and will keep you going in most of the project situations. We will explain Document Object Model (DOM) and Simple API for XML (SAX), and differences between them that will help you decide which to chose for your application.

Next we describe core standards of XML-based Web Services – Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI).

Peer-to-peer communication on Interned may be secured by widely used Secure Socket Layer (SSL) or Transport Level Security (TLS).

Java and Microsoft .NET technologies warrant special consideration. Competition between them and their ability to inter-operate will shape the landscape of Enterprise IT Architectures for years to come.

Chapter 9 is devoted to Java technology. Chapter 10 describes fundamentals of Microsoft .NET.

In Chapter 11, we provide a placeholder for 'other' or 'also' important for the Enterprise IT Architecture technologies, or just some fundamentals of computing that we did not have a heart to leave out.
We discuss hardware types, storage devices, Storage Area Networks (SAN) and networking, as a context for the layers of software. We identify major categories of software, starting from Operating Systems, their types and history.

We explain concepts of Programming Languages and cycle of program construction from the source code to the execution.

Chapter offers observations on Database Management Systems (DBMS), Transaction Processing Monitors (TPM), Application Servers and Middleware.

Open Source Movement provides a possible feasible alternative to the strictly commercial products in the Enterprise. We explore variety of Open Source models and products.

Chapter 12 brings together all the pieces of the Software Architecture knowledge base discussed earlier.
We focus on some core patterns in building the Enterprise IT Architecture. We discuss common high-level architecture challenges by revisiting the *Architectura Sapiens* example application from the Part 1, and by applying our new knowledge from the Part 2. You are likely to encounter most or all of these challenges on your Enterprise project.

<<< … >>>