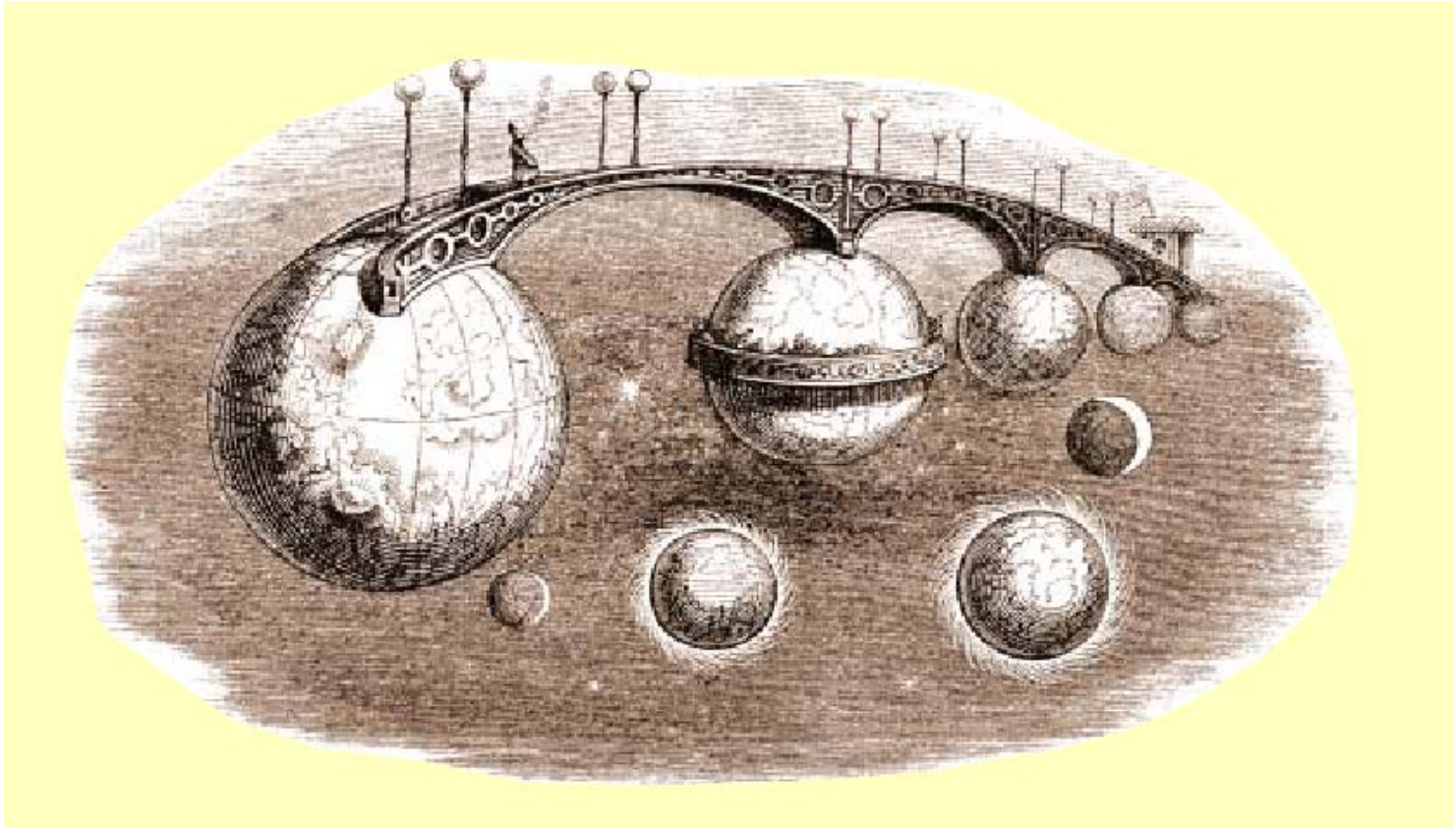
Contents and sample chapters for

Bridging the silos

Enterprise-architecture for IT-architects



Tom Graves

Tetradian Consulting

Published by

Tetradian Books

Unit 215, Communications House

9 St Johns Street, Colchester, Essex CO2 7NN

England

http://www.tetradianbooks.com

First published November 2008

ISBN 978-1-906681-02-9 (paperback)

ISBN 978-1-906681-03-6 (e-book)

Contents

An introduction 1
Basics – what is enterprise architecture? 4
Basics – the architecture process 14
Basics – artefacts and toolsets 15
Purpose – an overview 16
Purpose – business-driven architecture 24
Governance – an overview 25
Governance – roles and responsibilities 26
Governance – products 27
Framework – an overview 28
Framework – layers 34
Framework – primitives 36
Framework – composites 40
Framework – models 41
Framework – integration 42
Methodology – an overview 43
Methodology – preparation 51
Methodology – assessment 59
Methodology – solutions 61

Methodology – hands-off architecture 63
Completion – an overview 64
Completion – architecture artefacts 65
Completion – closing the loop 67
Glossary 68

Acknowledgements

By its nature as a 'conversion course' for IT-architects, this book draws extensively on existing frameworks and methodologies in common use in the IT-architecture space – in particular the Zachman Framework, and the Architecture Design Method (ADM) of The Open Group Architecture Framework (TOGAF). The author acknowledges that the originals are and remain the copyright of the Zachman Institute for Framework Advancement and The Open Group respectively. The adaptations described in this book were created in good faith to cover the needs of a broader purpose and broader market, and should not be interpreted as making any claims as to ownership of the original work.

Amongst others, the following people kindly provided comments and feedback on the early drafts of this book: Daljit Banger (White Knight Consulting, GB), Sally Bean (London, GB), Charles Edwards (ProcessWave, GB), Michael Ellyett (RHE, NZ), Nigel Green (CapGemini, GB), Ziggy Jaworowski (NSW DoCS, Aus), Kim Parker (Australia Post, Aus), Liz Poraj-Wilczynska (Brockhampton, GB), Chris Potts (Dominic Barrow, GB), Marlies van Steenbergen (Sogeti, NL), Peter Tseglakof (Australia Post, Aus). In particular, Michael Ellyett provided the innovative concept of 'hands-off architecture'.

Please note that, to preserve commercial and personal confidentiality, the stories and examples in this book have been adapted, combined and in part fictionalised from experiences in a variety of contexts, and do not and are not intended to represent any specific individual or organisation.

Registered trademarks such as Zachman, TOGAF, FEAF, ITIL, Macintosh etc are acknowledged as the intellectual property of the respective owners.

AN INTRODUCTION

An architecture for the enterprise

This book is about enterprise-architecture: not just IT-architecture, but *real* enterprise-architecture – the architecture of the enterprise as a whole.

Standard 'enterprise-architecture' descriptions, such as Zachman, TOGAF or FEAF (see *Resources* below), tend to imply that architecture begins and ends with IT. But IT architecture is just one part of a real enterprise-wide architecture, one subset of a much larger picture. Above all, the architecture needs to be *business-driven*, not technology-driven: the moment we forget that, we're in trouble straight away.

We need to remember, too, that there's a lot more to *real* IT than just 'computers and stuff'. A pencil is information technology.; a whiteboard is information technology; likewise the humble sticky-note. Creating conditions for meaningful knowledge sharing is information technology – so the layout of the office café can be a crucial part of an organisation's information technology. I'd doubt that any of those items would rate even a single mention in your existing 'IT-systems' models – yet they're all essential aspects of a full-scale enterprise-architecture.

So the bad news is that real enterprise-architecture requires us to deal with a much broader scope than the familiar comfort-zone of IT-architecture. We *do* need our architecture to address that full scope – the entire enterprise – if we're to create viable solutions that bridge across all of the silos.

The good news, though, is that as an IT-architect you're already well-equipped to tackle this:

- you're a generalist, used to looking at things from multiple perspectives
- you know how to build models at different levels, from abstract, or conceptual, to logical, physical and operational
- you know how to use models to explain complex concepts to many different audiences, from board-level to operations staff
- you know how to derive models from real 'as-is' situations

- you know how to model an architecture from a strategy or a required 'to-be' scenario for a system's future
- you know how to derive practical solutions from abstract models

All the essential skills you'll need, in fact. All that you'll need to add is a stronger grasp of the much broader scope of architecture at a true *enterprise*-wide level, and a willingness to place IT into a more realistic role and perspective within the enterprise.

That's what this book is about.

What's in this book?

In the previous book in this series, *Real Enterprise-Architecture*, I described a systematic model and methodology for enterprise-level architecture, based on the classic Group Dynamics five-phase model of the project life-cycle. But it seems in practice it can feel too abstract at first, especially for IT solution-architects: it can be too much of a jump to go direct to that level

So this book provides a 'conversion course' for IT-architects. It uses the same overall model as the previous book, with the same five phases or strands of the architecture process: but here the description for each strand is adapted from themes familiar to most IT-architects, such as Zachman, TOGAF, FEAF, ITIL, Agile, MSP and PRINCE2. The basic sequence is as follows:

- ***Purpose*** (see *Purpose – an overview*, p.16) – identifying the **business-purpose** of each architecture-iteration and the whole architecture – adapted from Agile, MSP and TOGAF
- ***People*** (see *Governance – an overview*, p. 25) – **governance** principles, responsibilities and procedures – adapted from MSP, PRINCE2 and TOGAF
- ***Preparation*** (see *Framework – an overview*, p. 28) – the **frameworks** used to keep architectural assessment information in context – adapted from Zachman
- ***Process*** (see *Methodology – an overview*, p.43) – the **methodology** for architecture assessment, solution-design and solution-implementation – adapted from TOGAF and Agile
- ***Performance*** (see *Completion – an overview*, p. 64) – the metrics and 'lessons-learned' processes used for **completion** and 'closing the loop' of the architecture cycle – adapted from TOGAF and PRINCE2.

Each section is split into bite-size chunks to apply straight away in your day-to-day work. Although there's a fair amount of theory, the keyword here is *practice*: the aim is to give you something that you can *use*. So each chapter includes examples and stories to place the ideas into a real-life context, with reference to other relevant resources. There's also a glossary at the end of the book, which should help in clarifying the broader meaning of some of the common terms used in whole-of-enterprise architecture.

But before we explore that sequence, we first need to address some basic issues, such as identifying what we mean by 'enterprise architecture'. So that's where we'll start in the next chapter.

Resources

- 🕮 Whole-of-enterprise architecture: see Tom Graves, *Real Enterprise Architecture: beyond IT to the whole enterprise*, (Tetradian, 2008)
- TOGAF (The Open Group Architecture Framework): see www.opengroup.org/togaf
- FEAF (Federal Enterprise Architecture Framework): see www.gao.gov/special.pubs/eaguide.pdf [PDF]
- Zachman framework: see www.zifa.org
- PRINCE2 (PRojects IN Controlled Environments): see www.www.prince2.org.uk
- ITIL (Information Technology Infrastructure Library): see www.itil.org.uk
- MSP (Managing Successful Programmes): see www.programmes.org
- Agile software / system development: see www.agilealliance.org and agilemanifesto.org

BASICS – WHAT IS ENTERPRISE ARCHITECTURE?

Summary

Enterprise-architecture: just two words, but to make sense of whole-of-enterprise architecture we need to be clear what we mean by each, as well as what's meant by the term as a whole. We also need to face the sheer scope of whole-of-enterprise architecture.

Details

Defining the enterprise

What is an enterprise? The FEAF document *A Practical Guide to Federal Enterprise Architecture* describes it as follows:

> [An enterprise is] an organisation or cross-functional entity supporting a defined business scope and mission.
>
> An enterprise includes interdependent resources – people, organisations and technology – who must coordinate their functions and share information in support of a common mission or set of related missions.

So far so good, but note the booby-trap here: 'enterprise' is *not* the same as 'organisation'. The *Practical Guide* warns:

> …it must be understood that in many cases, the enterprise may transcend established organisational boundaries – e.g. trade, grant management, financial management, logistics.

Rather like the dreaded 'org-chart', the boundaries of the organisation tell us almost nothing about the real bounds of the enterprise – the "defined business scope and mission". In practice, an enterprise is a *value-web* that includes the organisation's partners, suppliers, customers and all the other stakeholders within the

scope of that 'mission', with structures and relationships that may well be changing dynamically from minute to minute.

Throw a few other factors into the mix – such as outsourced business-critical functions, continuous '24/7' processes, or 'follow-the-sun' operations with interfaces shared worldwide across an entire industry – and the old notion that the organisation *is* the enterprise will seem more like a charming relic of a bygone age.

That's problematic enough in itself. But where it gets *really* messy is governance (see *Governance – an overview*, p.25). If the enterprise is more than the organisation, where are the boundaries of governance? Whose rules apply, and in which contexts? More on that later when we look at frameworks and governance.

Defining architecture

To understand architecture, let's look at the definition from the *Practical Guide*:

> [Architecture is] the structure of components, their interrelationships, and the principles and guidelines governing their evolution and design.

It's a good definition, though in some ways more for what it *doesn't* say than for what it does. It doesn't specify 'information technology', it says "components [and] their interrelationships" – leaving the definition open for the broader interpretation we're going to need.

But this still doesn't say much about the *nature* of that structure, and its "evolution and design". Most IT architecture is like the design of a single building: we describe the present state of the building, then what we want it to look like – the so-called 'future-state' – and some kind of plan or 'roadmap' to get from here to there. But at the enterprise level, architectural design is closer to town planning, or even the structure of an entire city. And at this level, it's orders of magnitude more complex. The world is emergent, unfolding, not static; there *is* no identifiable 'future-state'. We still need vision, a sense of future – or future*s* – but we need to work with something a great deal more flexible and fluid than a single, simple 'plan'.

Defining enterprise-architecture

To put this together, probably the best starting-point for a definition of enterprise-architecture comes from the Wikipedia:

> Enterprise Architecture is the description of the current and/or future structure and behaviour of an organisation's processes, information systems, personnel and organisational sub-units, aligned with the organisation's core goals and strategic direction.

The FEAF *Practical Guide* refers to this description as a 'strategic information asset-base'. But unlike the *Practical Guide*, the Wikipedia article also adds a crucial rider on scope:

> Although often associated strictly with information technology, [EA] relates more broadly to the practice of business optimisation in that it addresses business architecture, performance management, organisational structure and process architecture as well.

A much broader scope, explicitly aligned to a core *business*-question. When properly implemented, the 'strategic information asset-base' that EA manages is best described as 'the enterprise's knowledge of itself'. So let's use that as working definition:

Enterprise-architecture is a discipline through which an enterprise can identify, develop and manage its knowledge of its purpose, its structure and itself.

The business-*purpose* of that managed knowledge is to support organisational and enterprise change – what the enterprise is and does, where it's going, its choices in that journey. This suggests, in turn, that the business role for the EA unit would be as a support for an enterprise-wide programme-management office (PMO) – providing information about the constantly-changing form of the enterprise, and guiding ideas and strategies about the best use of the enterprise's resources.

But because, wrongly, EA is often "associated strictly with information technology", the usual place we find an EA unit is as a subset of IT governance – which is *not* a good idea. To understand why, we need to look at little more closely at the impact of scope.

Scope and enterprise-architecture

The real concern with scope is this: are we thinking wide enough to cover what the context actually *needs*? If we're not careful about our assumptions, things can go spectacularly wrong – yet because our assumptions prevent us from seeing what's happening, we become incapable of understanding *why* things are going wrong. If we don't have a wide enough scope, we'll pour more and more resources into the only part of the problem we can see – but things just keep on getting worse, and we can't see why, or how.

Sometimes even the most basic of reality-checks can get lost along the way. A real conversation from a Denver-Boston flight, way too many years ago.

"I'm a senior project leader on Strategic Defense Initiative", he'd said, with patriotic pride. "You know – our space-based anti-missile shield!"

I'll admit I stifled a bleak laugh. "How does it work?" I asked.

"It fires an X-ray laser at the target – using a nuclear bomb inside itself to generate the laser."

"A nuclear bomb? In space? Isn't that in breach of international treaty?"

"No-one cares about *that*", he said, airily: no further questions there…

"So, uh, how does it defend itself against possible attack?"

"It fires off the X-ray laser, of course." Scornful.

"I don't get it", I said. "Doesn't that mean its only defence – even against a dummy missile – is to blow itself up?"

Sudden silence. A long pause. "We hadn't thought of that", he said…

Far, far too often, the technology comes first, and middle and last as well. The scope ends up being defined back-to-front, with the core business-needs all but forgotten. In one of my recent projects, for example, the IT group demanded that the business should redesign all its processes to fit in with the limitations of their badly-botched CRM implementation. In the project before that, the team-lead insisted that his made-up UML use-cases *were* the business-requirements – he'd never bothered to ask the business what they wanted. As an enterprise-architect, I must admit I'm a bit fed up of having to sort out the resultant mess.

The painful fact is that whenever technology takes the front seat, we end up with an expensive mess that doesn't *quite* do what's needed and has subtle yet serious side-effects. For which the only solution offered is yet another inadequately-thought-through technological 'fix', which also doesn't solve any actual problem but costs yet another lost fortune. And all too often, IT-centric 'enterprise architecture' has been just one more example: round and round on the same horrendously expensive cycle, getting nowhere faster and faster…

So the key concerns about scope are these:

- when dealing with any real-world issue, any restriction on scope and 'solutions' will always cause *big* problems
- it's all too easy to artificially restrict scope by staying within our comfort-zone – sticking to what we 'know', to what we think we're certain of, to what we can control

- with a narrow focus, we have no means to see *why* things are failing – and we make things worse by trying to force things to fit our too-limited assumptions

To avoid classic IT-centric disasters, we need to keep challenging our assumptions about scope. Which is rarely easy or comfortable, but we *must* do so, because in the real world, every item is connected to *everything* in some sense or other. Any restrictions we place on scope are an operational convenience, not a 'fact'. So what we need to remember is this:

There's always another view, another way to do it.

And that other way may well be better – more efficient, more effective – than what we're doing now.

> I started my professional career as a typographer, moving sideways through computer-based typesetting to more mainstream IT. I was fortunate to work with some of the best consultants in pre-press, and one incident comes to mind here.
>
> At a directory-publishing operation, the programmers were building an application to go direct-to-plate from the mainframe database. They'd struggled for days with a peculiar one-off case that wrecked one section of output at the end of a very expensive thousand-page run. And there seemed to be no way to get the system to do it right.
>
> My colleague listened to all the lamentations, looked at the galley-prints on the cutting-board, and grinned. He reached into his briefcase, pulled out a scalpel, and with a few deft strokes re-arranged the offending symbols into the required order. The whole thing took a matter of seconds – and the result was exactly what was needed.
>
> The programmers looked on in horror. "We can't do that with code!", said one. "That's *cheating*!"
>
> "Cheating?" said the consultant, with a wry smile. "Getting the job out the door right, on time and on budget, is 'cheating'?" A few crestfallen faces; embarrassed shuffling of feet. "Getting the code right is important; but it's easy to forget the *why* for the code in the first place." He placed the scalpel on the desk. "Keep it", he said. "Label it 'for emergency use only', if you like – but there are times when it's the best tool we have!"

This is another reason why it's dangerous to place enterprise architecture under the 'IT governance' banner. Viewing the world through an IT-only lens means that, almost inevitably, we'll try to use IT methods to 'solve' problems for which IT isn't suited. To make sense of the whole, we need to start our overview from a much higher viewpoint than the detailed operations-level of one IT-centric subset in the enterprise – which is where we'd find conventional 'enterprise-architecture'. So we need to anchor our scope on one simple fact:

Every activity begins and ends with a business purpose.

No matter where we set our scope - even from right at the top of the enterprise – we'll still get lost if we lose sight of the 'why' of the enterprise. Without that anchor, the pressures of day to day business can push us a long way off track before we've had a chance to notice it – and once we do become aware of what's happened, it can take a long time get back on track. Even in the fine detail of moment-to-moment operations, we need some way to keep track of the business-purpose of what we're doing.

A painful personal example. As a writer and typographer, I moved into typesetting to gain more control over the production of my books. But I got sidetracked into running a typesetting business, to pay for the typesetting-system; and thence into micro-computers, to get more value out of our machines and make them more available to others.

Like so many people, I got hooked on code – that ever-elusive goal of getting the wretched machine to do what I *wanted* it to do! Hence rather too many all-night sessions, coding and debugging and the like – all of it in low-level assembly-language, in those days. Not far off a coffee-addiction. And *way* too much stress all round.

A year or so later, I was running what had become probably the first real micro-based desktop-publishing operation in Britain – half a decade before the Macintosh and PageMaker. We wrote code to typeset magazines, partworks, bank rates-tables, even crossword-puzzles. A tiny operation, based in the back end of Britain, doing innovative, world-leading work for some of the largest companies in the country. It was a heady time.

But what I *didn't* do was write books. I'd lost track of the original business-purpose.

It wasn't till some years later, sifting through the metaphoric wreckage of what had once been my own company, that I realised what I'd done, and what I'd lost. Others had long since taken over in my previous profession: it was way too late to go back. All I could do was go forward, to start again in another field of interest – and keep reminding myself to remember the real business-purpose in whatever I did.

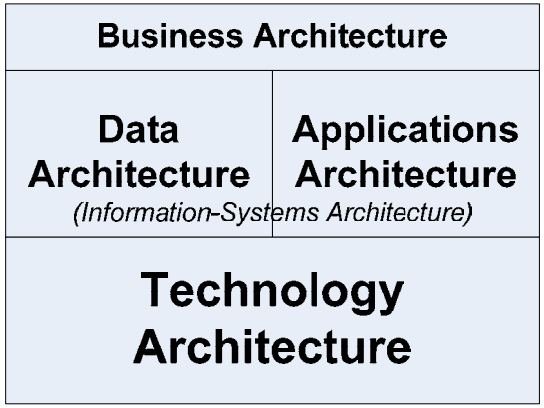
The inverse is also true: if we're starting from the high-level 'helicopter view' of strategy and the like, we have to remind ourselves of what's happening and what it's like right down in the low-level detail of operations. It's all the same continuum – the same body of knowledge, 'the enterprise's knowledge of itself' – that architecture needs to manage on behalf of the whole enterprise.

That's where IT-architects have a great advantage. Unlike most strategy-folks at one end of that continuum, and most operations-folks at the other, we're used to dealing with entities at every level. But whilst we're likely to be good at managing depth, as IT-architects what we're *not* likely to be good at as yet is breadth: a broad enough grasp of the *whole* of the enterprise, in all its complexity. So that key difference between *IT*-architecture and real *enterprise* architecture is what we need to turn to next.

Broadening the scope

In conventional IT-centric architecture, the world of the enterprise is divided up into four neat packages – business, data, applications, and technology – organised into a neat vertical hierarchy. For a top-down view of the enterprise, we start from 'business' and strategy; for a bottom-up view we start from the low-level details of technology.

There are a few variations: TOGAF's hierarchy is as above, whilst FEAF places applications above data, and others add an extra 'information' layer between 'business' and 'data'. But in essence this, we're told, is the complete scope of enterprise architecture – all that we need to work with, all that we need to know:



The IT-architecture hierarchy

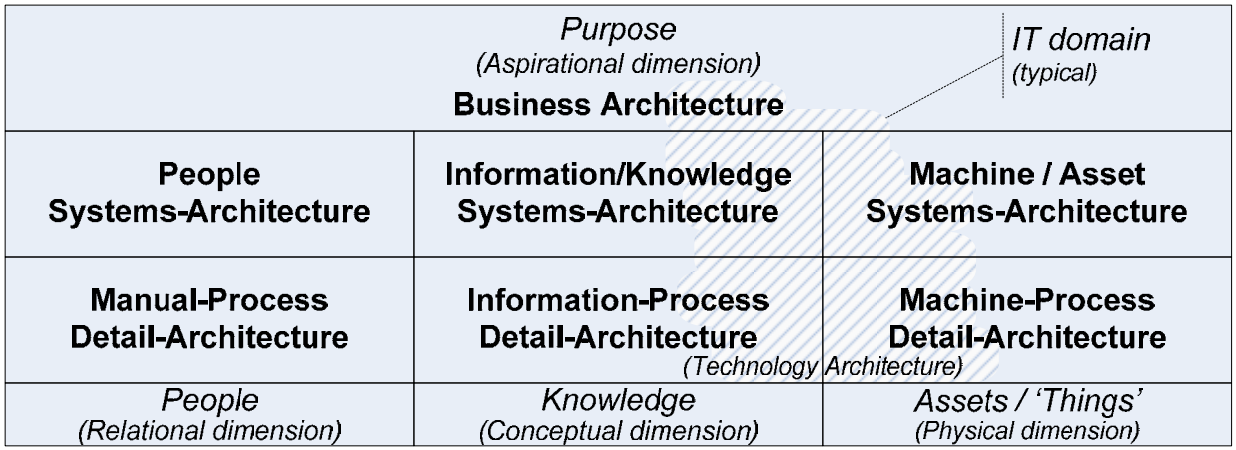
Which is, bluntly, just plain stupid. To ask one really simple question, where do people fit within this scheme of things? Nowhere, is the short answer. Yet ITIL, the most commonly-used IT service management framework, insists that people are at the core of all service-management issues. Even from an IT perspective this 'standard' hierarchy is too incomplete to be any real use. So what's gone wrong? And what can we do about it?

The problem stems from enterprise-architecture's history. It began with piecemeal attempts to rein in some of the more rampant computer-based chaos, and later gained an expanding breadth of coverage as people started to realise that everything was dependent on everything else. Eventually it became clear that this 'IT-architecture' needed an enterprise-wide scope – at which point some idiot started calling it 'enterprise-

architecture', as a shorthand for 'enterprise-wide IT-architecture' But it's misleading, because *the focus is on the technology, not the enterprise*.

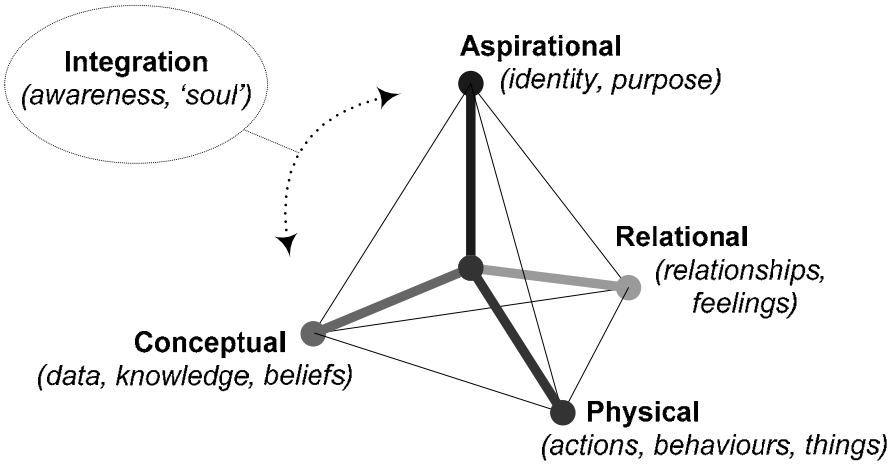
It's true there's an increasing awareness that IT-strategy needs to be business-driven: but in most cases 'business-architecture' is just a label for 'everything not-IT', a kind of random grab-bag with low-level business-processes all jumbled together with high-level business strategy and everything in between. And it's still so technology-centric that the TOGAF Enterprise Edition methodology, for example, assigns more than six times the number of assessment-steps to technology than everything else put together.

In short, it's not *enterprise*-architecture at all: it's an IT-centred mess. TOGAF gives us its fixed hierarchy of business, data, applications and systems-technology; but the *real* world includes people, and the knowledge held in people's heads, and cars and trucks and printing-presses and fork-lift-trucks and other machines that don't have a scrap of IT in them, and a great deal more besides. In other words, a *real* enterprise architecture needs to describe a world that looks more like this:



A more realistic hierarchy for TOGAF

Another way to portray this is to show the major dimensions – purpose, relations, knowledge and physical 'things' – in a tetrahedral relationship.



The tetradian: rotating the hierarchy

Rotating attention between each axis of this *tetradian* helps to keep the perspective in balance: This is true even for an IT-centric organisation such as a bank or an insurance firm; far more so for manufacturers and retailers and others, such as most government departments, for whom IT is simply an enabler, not the core of the business.

A IT-centric view of the enterprise, TOGAF-style, is simply one possible *view* into the broader scope represented by the whole enterprise. But it's only one of *many* possible views, all of them equally valid. So please, let's hammer this point home:

Enterprise architecture covers the whole enterprise

IT is one small subset of the enterprise

IT-architecture is one small subset of enterprise-architecture

Because as enterprise-architects, if we ever forget that, we're in deep trouble.

Application

- What is your current definition of 'enterprise architecture'?
- What is the current scope of the enterprise?

- What is the current scope of enterprise-architecture?
- Where – if at all – is enterprise-architecture sited in your organisation?

Resources

- FEAF: *A Practical Guide to Federal Enterprise Architecture*: see www.gao.gov/special.pubs/eaguide.pdf [PDF]
- ITIL: see www.itil-officialsite.com
- TOGAF layers: see www.opengroup.org/architecture/togaf8-doc/arch/toc.html (chapter 'TOGAF as an Enterprise Architecture Framework')
- Wikipedia summary on enterprise-architecture: see en.wikipedia.org/wiki/Enterprise_architecture

BASICS – THE ARCHITECTURE PROCESS

Summary

The architecture process has two distinct phases: assessment against requirements, in order to derive an overall design; and oversight of implementation, to ensure the final result does match both the design and the requirements. This applies to every level of structural change, from a single project to the whole enterprise. In whole-of-enterprise architecture, a key complication is that the world moves on whilst implementation takes place: design for flexibility is a must.

Details

[[see published book for further details]]

BASICS – ARTEFACTS AND TOOLSETS

Summary

Creation and maintenance of documents, models and other artefacts will form a key part of every architect's working life. You'll need an appropriate toolset with which to do this.

Details

[[see published book for further details]]

PURPOSE – AN OVERVIEW

Summary

The purpose of architecture is to support the changing needs of the business. At the whole-of-enterprise level, a simple strategic plan will never be enough – it needs to cope with a much higher degree of dynamic complexity. And whilst IT-architecture often talks in terms of 'engineering the enterprise', a more useful metaphor here is that of the 'living enterprise' – purposive, pro-active and *self*-adapting to change.

Details

Architecture on purpose

One phrase we'll often hear in conjunction with enterprise architecture is 'business/IT alignment'. The idea is that somehow the architecture should bring the aims of each side of the divide into alignment with each other, into agreement with each other.

IT people being who they are, of course, there's a hidden tendency to assume that it's the business that should change, to align itself with the needs and aims of whatever systems the IT group wants to provide – or all too often, whatever the vendors want to sell… And business people being who *they* are, they're not likely to be happy about this – *at all*. Which is not surprising, because they're right: every activity begins and ends with a *business* purpose. In that sense, business always comes first – and *must* always come first.

More to the point, there can be no separation here, no 'us and them' – and especially no 'us *versus* them'. There's a natural tendency amongst IT folks to see themselves as 'special and different', but their role, in essence, is that of just one more support-function for the overall enterprise. The moment a sense of separation is introduced, the whole enterprise is put at risk.

> That supposed separation – and likewise the somewhat arrogant sense of 'special and different' – puts IT's place within the enterprise at risk, too. Many IT departments tend to view themselves as a sort of 'insourced supplier' to the enterprise, with separate IT strategies, separate IT-governance, and so on. But this is not a wise tactic for IT – kind of

like sawing off the branch you're sitting on, in fact – because it invites the rest of the business to view IT as an *external* supplier, and hence a suitable target for *out*sourcing. In which case, bye-bye to many – most? – of the IT jobs; and hello to total confusion all round in a year or two, most likely…

In short, if you want to be part of the enterprise, *be* part of the enterprise – don't present yourself as separate!

The other critical point here is around scope and scale. It's common for IT to describe its concerns in terms of planning, or at best in terms of strategy for example, one toolset-vendor recently re-badged its enterprise-architecture offering as 'IT strategic planning'. But this too can be a mistake. at the whole-of-enterprise level, because the scope is too large and too complex for anything as predictable as a plan. To see why this matters so much, it's worthwhile to take a brief detour through a matter of metaphor – because the way we choose to *view* the enterprise also delimits what we can do *in* the enterprise.

A matter of metaphor

The key distinction we need to note here is between two different metaphors:

- the enterprise as *machine*
- the enterprise as *living organism*

The first is the classic view of the enterprise: a kind of 'machine for making money', in the commercial context. It's the metaphor that underpins Zachman's assertion that the role of enterprise architecture is 'engineering the enterprise'; further back, it's the same metaphor beneath Frederick Taylor's century-old notion of 'scientific management' that gave birth to Henry Ford's autocracy of the assembly-line. Inspired by Ford's apparent success, state-Communist regimes applied the same metaphor to the social realm, with an almost religious faith in the desirability, efficacy and achievability of the 'five year plan'. We might note, though, that not one of those regimes has withstood the test of time …

And likewise, it seems, the 'five year plan'. A week or so ago, at a seminar run by one of the EA toolset-vendors, a speaker asked for a show of hands as to how many IT executives still had a five-year plan. Five years earlier, perhaps even three years, perhaps even two, pretty much everyone would have had a predefined plan. But this time not a single hand went up. Not *one*. Interesting…

Much the same reason, I'd guess, as to why another colleague argues that the best strategy for IT is to *not have* a strategy – or no static strategy, at least.

So what's changed? Nothing, really. It's just that people have at last begun to realise that the five-year plan doesn't work – more to the point, that it *can't* work in the real world. It never *has* worked – that's the bleak irony here…

There's no doubt that the machine-metaphor does sort-of work in simple contexts with minimal variance and minimal change – the kind of problems we used to deal with, for example, in the monolithic world of old-style mainframes in which Zachman's 'enterprise architecture' first emerged. But that isn't the world we deal with now in most enterprise architecture: what we need instead is a metaphor that *can* cope with complexity. Which is where the metaphor of the 'living enterprise' comes into the picture. Which in turn is the reason for a renewed emphasis on business-purpose.

There's a subtlety here that turns out to be extremely important: a machine may be built *for* a purpose, but it doesn't *have* a purpose of its own. Any purpose the machine may have must come from *outside* of itself: it has no intrinsic purpose *in* itself. Hence armies of external apparatchiks – otherwise known as 'consultants', in the business world – to do the thinking for the machine, and to provide its purpose. Yet the Taylorist trap is that this comforting illusion of 'control' comes at a cost, because response to change slows to a crawl. Which is not a survival tactic in a rapidly-changing world…

Contrast this with the metaphor of the 'living enterprise'. The idea has been around for quite a few years now, but has gained business momentum – in Britain at least – through the activities of Charles Handy and others on the RSA's 'Tomorrow's Company' group. (The RSA – or Royal Society of Arts, Manufactures and Commerce, to give it its full title – is the world's oldest formal business organisation, founded way back in the seventeenth century.) And it's arguable that the living-enterprise metaphor is the only one that works well at the whole-of-enterprise level.

The key concept is that the enterprise is viewed not as a collection of discrete, separate parts, but as a web of interdependent services, all sharing a common aim. None of the parts is inherently 'better' or more important than any other: each has their own role to play – so management provides 'management services' to the whole, HR provides 'people services', operations provide 'production services', and so on. In the living organisation, 'the brain of the firm' – to use the term coined by cyberneticist Stafford Beer – is not something separate, but is distributed throughout every aspect of the enterprise.

In this context, 'service-oriented architecture' is no longer just an IT-industry buzzword, but an accurate description of the entire enterprise. Stafford Beer's Viable System Model has been proven in practice for half a century as a successful framework to structure the information-flows – the metaphoric 'nerve-system' - of large, complex enterprises, up to the scale of entire country in one case. 'Viable system' design principles, such as recursion and reflexion, can also be used for other whole-of-enterprise services such as infrastructure, energy-supply, quality-management, strategy,

security, business-ethics and the like – the living-enterprise equivalents of skeleton, veins, arteries, endocrine system and so on – though we'll explore that in more depth in a later book in this series.

The whole is not just the sum of its parts – as it is in the machine-metaphor – but is *more* than the sum of its parts: and every part *matters*. And we see the same principles at work in other business-themes such as Six Sigma and Total Quality Management. When 'the machine' becomes *self*-directing, 'self-actualised', responses can become fast enough to cope with the real-time complexities and confusions of the real world. And the sense of purpose – an *internalised* sense of purpose – is what drives those responses, ensuring that each can be appropriate to context and need.

That clear sense of purpose also provides resilience, it seems. LloydsTSB Bank was an early adopter of the 'living enterprise' metaphor – and was one of the few large British banks to survive the 2008 'credit crunch' relatively unscathed. Might be a useful lesson there for others in the finance industry and elsewhere, perhaps…

In the machine-metaphor, we try to take control of the enterprise, through predefined plans and strategies. The reality is that it doesn't work: all manner of proven reasons as to *why* it can't work, but the fact is that it doesn't. But in the living-enterprise metaphor, we don't even *try* to plan: instead, we provide *direction*, through purpose. In the same way, so-called 'city-planning' isn't about *planning* as such, but about direction, about purpose. At every scale, in every decision, every service, the purpose *defines* the enterprise – and that in itself is what makes all the difference.

Coping with complexity

As enterprise architects, charged with tackling so large a scope, how on earth *do* we cope with all that complexity? It's actually a lot simpler than it looks, as long as we remember two key points:

- don't try to do it all in one go
- don't try to control the complexity – work *with* it, not against it

Although some of the lesser-known IT-architecture frameworks such as Sogeti's *DyA*, or 'Dynamic Architecture', do provide some help in this, most of the common frameworks fail on both counts – another reason why they can become recipes for expensive insanity at the *enterprise* level. John Zachman, for example, insists that everything about the entire enterprise should be recorded in "excruciating detail" – a worthy goal, perhaps, but not a realistic one, not least because it takes so long that the detail is out of date before we're even a tenth of the way through.

In a sense, though, Zachman is right. We *do* eventually need that level of detail – but *only* where we need it, not absolutely everywhere. We'll come back to this again when we look at methodology, but for now it's useful to compare two different approaches to understanding any large system: *analysis*, and *holism*.

Analytic assessment and holistic assessment

Conventional *analysis* breaks a large system into manageable chunks – 'eating the elephant one bite at a time' – by splitting each chunk into parts, and those into smaller parts, and so on. The emphasis is on identifying the detail of each item and its individual components. But the catch is that once we get down into the kind of low-level detail we need to solve real-world problems, it's all too easy to lose track of where each part fits within the whole. It's like if we cut up a photograph into small pieces: we're left with a jigsaw-puzzle of very fine detail, but no 'big-picture' to tie it all together. And because there's nothing to tie all the pieces together, analysis *demands* stability: if the world moves on whilst we're working on it, we're lost.

By contrast, an *holistic* approach is more like cutting up a holograph. Each tiny piece of the picture may seem blurry and indistinct, yet each maintains within itself a pattern of the whole. So although we still do assessments that *look* like analysis – often going all the way down to Zachman's 'excruciating detail' – the real emphasis is on *patterns*, and on connections *between* items rather than the items themselves. Each dive down in to the detail also provides a bit more certainty about the nature of the whole. And because everything is linked together from the start, we don't need certainty or stability: we *can* work on a changing world.

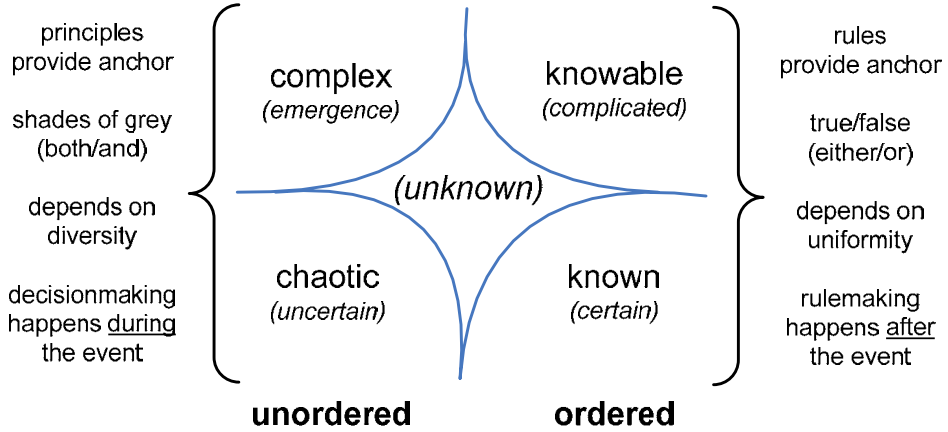
To make it work, we *start* with a 'big-picture' view. To put it bluntly, we make it up: it's an invention, nothing more than that. But we create it from what we can see of those aspects of the enterprise that don't change over time – particularly its guiding principles and values. These form the skeleton or backbone of the framework for our 'holograph', to which we add more and more detail as we go.

How does this help us tackle what would otherwise be an overwhelming complexity? One answer is that we don't set out to create and impose some kind of top-down 'city master-plan': instead, we build the hologram iteratively, from every possible perspective, engaging the views and opinions of people at every level and in every domain. We may well need some kind of unifying design to start with – see 'The architecture cycle' in *Methodology – an overview*, p.45 - but as the architecture matures, we're probably better to adopt a more 'hands-off' approach – see *Methodology – hands-off architecture*, p.63.

The other answer, perhaps, is to use models that explicitly address the complexity – rather than shy away from it, as in the usual 'control'-based approach to enterprise architecture. For this, one of the most valuable models in our toolkit is one originally developed at IBM by Dave Snowden and others, called Cynefin.

Cynefin and complexity

Like all good models, Cynefin is deceptively simple. Given a context that seems unknown, it says, we have four ways to work with it: assume it's known; analyse it into something knowable; find some way to work *with* the emergent complexity; or accept the implicit uncertainty of a unique 'market of one'.



Cynefin model

Each domain has a different decision-making style:

- 'known' domain: based on predefined *rules*
- 'knowable' domain: based on *analysis*
- 'complex' domain: based on *guidelines* and heuristics
- 'chaotic' domain: based on *principles*

In a sense, these form a hierarchy. At the simplest rule-based level, we need know little or nothing about business-purpose; but as the world we deal with becomes more complex and chaotic, purpose matters more and more. Hence, again, why clarity on and alignment with business-purpose becomes ever more important as we expand the scope and complexity of the enterprise architecture.

Decision-making is fast in the 'known' and 'chaotic' domains, but require radically different levels of skill to be reliable; decision-making is slower in the 'knowable' and 'complex' domains, but at least we have a better idea of how we get there! More on this later when we look at mapping capabilities and decisions within the architecture-framework – see *Framework – primitives*, p.36.

Difficulties with dynamics

At the highest level of business-purpose, the Vision – as it's called in ISO-9000:2000 – identifies the core values and principles that define the enterprise. These never change – or perhaps more to the point, they're so much the core of the enterprise that if they *do* change, it's no longer the same enterprise.

> One of the classic examples of this is the 'HP Way', the original guiding principles for Hewlett-Packard. When those principles were in effect abandoned by a new management, quite soon after the founders' deaths, the company went into a steep decline, from which it seems still yet to recover.

But those core values are the only things that don't change in the enterprise, and in the enterprise architecture. Everything else *does* change. So we perhaps again need to hammer this point home:

The world is not static; there is no 'future state'

Talking about 'as-is' versus 'to-be' is fine, as far as it goes; but trying to define a 'future state' isn't, because there is no possible 'steady state' – especially not at the scope of whole-of-enterprise architecture.

One other trap here is the tendency to regard the organisation as '*the* enterprise'. It isn't. It's true that the organisation is an enterprise in its own right, but the *real* enterprise – the one we need to identify and model in our enterprise architecture – is a much more fluid, dynamic entity: more like 'enterprise*s*' plural, really. Remember that definition earlier: an enterprise can be *any* kind of grouping, of *any* size or boundaries, that share a common purpose within a common scope. It can be as small as a single work-team; it can be as large as a industry-wide consortium; ultimately, in certain contexts, it might well include the entire world. It's only at the very top that the purpose is static; over time, everything else will change scope and boundaries and details of purpose, merging, demerging, in every possible combination. *Everything*. That's a lot of flexibility, a *lot* of change, all the time – but that's what real enterprise architecture demands.

Yes, it's complex – enormously so – and often with nightmare problems with versioning and the like. But it *is* doable, as long as we remember those two points: don't try to do it all in one go – because we can't – and do use that holistic overview to manage the dynamics of complexity.

Application

- How do you ensure an emphasis on business-purpose in your architecture?
- Who defines business-purpose for your architecture?
- How do you ensure business engagement in architecture development and use?
- How do you manage complexity and dynamics in your architecture?
- How do you ensure a balance between analysis and holistics in your architecture?

Resources

- FEAF: *A Practical Guide to Federal Enterprise Architecture*: see www.gao.gov/special.pubs/eaguide.pdf [PDF]
- TOGAF (The Open Group Architecture Framework): see www.opengroup.org/togaf
- RSA 'Tomorrow's Company' group: www.tomorrowscompany.com
- DyA (Dynamic Architecture): eng.dya.info/Home/
- Cynefin: Wikipedia overview at en.wikipedia.org/wiki/Cynefin; more detail at Cognitive Edge: www.cognitive-edge.com
- The 'HP Way' principles: see www.hpalumni.org/hp_way.htm

PURPOSE – BUSINESS-DRIVEN ARCHITECTURE

Summary

All architecture begins with a business-purpose, a business question to be addressed. These 'business drivers' can take many different forms, from 'top-down' strategy or 'bottom-up' impacts of real-world incidents, to routine replacements and re-engineering or resolution of complex 'pain-points' for the business.

Details

[[see published book for further details]]

Horizontal drivers

Top-down drivers

Bottom-up drivers

Spiral-out drivers

GOVERNANCE – AN OVERVIEW

Summary

Governance addresses the people-issues around the 'who' and 'why' of enterprise-architecture: who does what, who is responsible for what, and why things should be and have been done. The system for governance provides formal structure for the processes that manage these issues.

Details

[[see published book for further details]]

Architecture and the enterprise

Governance drivers

Governance styles and standards

Governance in the multi-partner enterprise

GOVERNANCE – ROLES AND RESPONSIBILITIES

Summary

The 'who' issues of architecture-governance revolve around roles and responsibilities: who is responsible for what, in what sequence, and why. Keeping the business-purpose in mind at all times is essential here – as is an awareness that people need to be respected *as* people, and not as machines subject to governance 'control'.

Details

[[see published book for further details]]

Roles and responsibilities – an overview

Identifying the stakeholders

RACI, CRUD and other matters

Partitioning the responsibilities

GOVERNANCE – PRODUCTS

Summary

Much of governance – in PRINCE2 especially – is monitored through 'products', the artefacts that record the outcomes of processes. This section summarises the various types of architecture products and their roles.

Details

[[see published book for further details]]

Documents and other 'products' – an overview

Guidance products

Management products

Reference products

Metrics products

FRAMEWORK – AN OVERVIEW

Summary

The framework provides a means to categorise and cross-reference the various business-entities identified in the architecture. Because the framework also determines the business meaning of each entity, the well-known IT-centric frameworks such as Zachman and the FEAF Reference Architecture can be problematic for whole-of-enterprise architecture: at the root-level especially, the framework *must* always cover the whole scope of the enterprise, not just its IT.

Details

Taxonomy, ontology and other tortuous terms

The framework describes the 'what' of the architecture. And of all the frameworks for enterprise-architecture, Zachman's is perhaps *the* best-known artefact in the entire field. It consists of six horizontal rows or layers, representing different strategic or tactical perspectives (see *Framework – layers*, p.34); and six vertical columns, representing key categories of entities (see *Framework – primitives*, p.36). Between them they form thirty-six cells, each of which is supposed to represent a single type of indivisible 'primitive' which can be used to describe entities for compliance and re-use in the architecture.

	What	*How*	*Where*	*Who*	*When*	*Why*
Scope						
Business						
System						
Technology						
Implement						
Operations						

The Zachman framework

Looking at Zachman, it's easy to form the impression that it's a list of 'things' the business needs: data, processes, applications, locations, physical objects such as servers, routers and networks. Tick off the boxes, drop the items in the right pigeonholes, and that's your enterprise-architecture pretty much done, isn't it?

Uh… not quite… More like a *long* way from done, in fact, because the framework is much more than a set of boxes: it's what determines *meaning* in the architecture. The key issues here include two tangled terms: taxonomy, and ontology.

Taxonomy – literally 'the naming of arrangement' – identifies the ways in which items are organised and categorised within the architecture. It also specifies and determines the recognised or 'legal' relationships *between* items. Zachman's framework provides a useful overview-taxonomy for architectures, but in itself it's all but meaningless – literally so – because it provides no real ontology for that framework.

Ontology – literally 'the study of meaning' – expands on or links to a taxonomy by assigning agreed meanings to each of the items within it. There are a variety of forms this might take:

- *glossary* or 'controlled vocabulary' – a list of agreed terms applied to those items
- *thesaurus* – a list of agreed relationships between those terms
- *schema* – a list of attributes and attribute-types associated with each item-type, and their relationships and dependencies
- *metamodel* – a variant of a schema which includes visual representations of how the items and their relationships should be portrayed

Ideally, a glossary and thesaurus should be structured around a formal syntax, though it's not always essential (see 'Glossary and Thesaurus' in *Completion – architecture artefacts*, p.66). Schemas and metamodels should always follow a formal syntax, as they determine the structure of the architecture repositories and other stores (see 'Repositories' in *Completion – architecture artefacts*, p.66).

All of these items need to be maintained under explicit governance. In the architecture methodology, the main governance activity occurs in the Preliminary Phase prior to any architecture-cycle, or in regular formal reviews of the entire architecture (see *Methodology – preparation*, p.51). Context-specific reviews may also take place during the 'lessons learned' stage in the final phase of the architecture-cycle (see 'Phase H – review lessons-learned' in *Methodology – solutions*, p.62).

Primitives versus composites

In the architecture repository especially, metamodels tend to be stacked on top of each other in layers. In part this reflects the different views of the enterprise, as in Zachman's framework, and the different ways in which the various models and entities are used. But at the root, the key distinction here is between two different architectural functions, described by Zachman as *primitives* and *composites*.

The simplest analogy is that primitives are the atoms or root-elements of the architecture. Like molecules, the composites are architectural solution-components constructed from any required combination of simpler entities. Composites may often be layered, in the same way as a complex molecule such as DNA is itself structured from simpler molecules such as RNA. As with DNA, re-structuring a composite, or changing some of its components, will change its function, perhaps to better suit a changed environment. Yet ultimately it needs to be possible to resolve every composite all the way down to its distinct elements, its root-primitives. This is crucial, because, as Zachman explains:

Primitive models are architecture.
Composite models are implementations.

Every composite is a pre-packaged 'solution' for something. Composites guide solution-architecture: for example, the 'Solutions Continuum' in the TOGAF reference framework is made up of layer upon layer of well-defined composites as 'Solution Building Blocks'. But it's still only *solution*-architecture – the processes that happen *after* we've done the core assessment work for our *enterprise*-architecture.

This isn't trivial. If we can't resolve all the way down to root-primitives – TOGAF's 'Architecture Building Blocks' – we're stuck with the solutions we already have, which may not fit a changed context at all. And if we don't have the right set of primitives at the root-level, we'll be unable to *see* new solutions, or to have any way to rethink what's going on in our enterprise: the taxonomy and ontology define meaning, and hence what is visible, what is deemed 'real' and relevant, and what is not. In his seminars Zachman does take care to portray his framework as covering more than just IT; but most toolset-based implementations of the framework don't – with unfortunate results.

> I'm reminded here of a children's-television series of many years ago called *Sapphire and Steel*. In the story, each element was personified as a kind of superhero in a stereotypic war between good and evil. Each character reflected the attributes of its element: Mercury could flow around obstacles, Lead had enormous mass, and so on; "transuranic elements must not be used where life is present", intoned the titles voice-over. However, half the real elements were missing – lack of budget, perhaps? – but the list of 'elements' did include the eponymous Sapphire and Steel. Often wondered how many kids were confused about chemistry as a result of that little bit of artistic licence…
>
> Sapphire and steel are compounds, of course – composites, not primitives. Anyone trying to build a real-world chemistry on the assumption that sapphire and steel are elements is going to come unstuck; and they'll find their options seriously limited if they're missing great chunks of the Periodic Table. But that's pretty much what happens whenever someone tries to do real enterprise-architecture with a conventional IT-centric framework and toolset. Every example I've seen so far is a muddled mixture of composites such as 'Process' or 'IT Application' masquerading as primitives, whilst half the root-primitives we *do* need for whole-of-enterprise architecture – items such as 'physical event', or 'social-network location', or even 'person' – are either in completely inappropriate places in the taxonomy or missing altogether. No wonder it's such a mess…

I'm occasionally accused of fanaticism on this issue, but it really does matter, because our flexibility of options depends on the precision of the metamodel: the closer we can get to true primitives, the more flexibility we have. Most of the time we'll work well above that layer of primitives, but there are a few times – such as planning for disaster-recovery, or for major legislative change – when we really *do* need to get right down to the roots and rethink things from scratch.

Each cell in Zachman's framework represents a class of primitives, whilst composites straddle horizontally across multiple cells of the framework. There are no vertical composites as such: instead, vertical structures such as logical-to-physical data-maps are more accurately trails of derivation or implementation.

The layers in Zachman are usable almost 'as-is' for whole-of-enterprise architecture, because they represent different views and time-frames in the architecture (see *Framework – layers*, p.34). The columns, however, do

need much more work to be usable: not only are some of the category-assignments wrong, but in effect there's an entire dimension missing, especially for cells for the lower layers (see *Framework – primitives*, p.36). It's only when we've sorted that out that we have a stable foundation on which to map the layers of composites for solutions – in particular, the intentional 'incompleteness' of components, especially in the higher layers of the framework, which supports their re-use in differing contexts (see *Framework – composites*, p.40).

Entities, properties, relationships and models

The most visible result of the framework is the set of models that many people think of *as* 'architecture'. In essence, though, a model is a description of ideas about the structure of relationships between various types of architectural entities – for example, the 'things' implied by the pigeonholes in the Zachman framework. Each type of model will contain three categories of information:

- *content* – the nouns of the model, providing references to the entities or 'things', such as data-records, or trucks, or business events, or locations, or whatever, sometimes with additional descriptive details of attributes or properties of each entity
- *context* – the adjectives of the model, providing further information or 'metadata' that would place those entities (or the records of those entities) within a broader context, such as the person responsible for that category of entity, or the date and time that the record was last changed
- *connections* – the verbs of the model, providing information about how the entities relate with each other, often literally in the form of a verb such as '‹implements›' or '‹aggregates›'

In a sense, models *are* relationships: each standard model-type provides a standardised means to describe relationships in a structured way. We describe that inner structure of the model-type in a schema or metamodel; in turn, the structure for that would be described by a meta-schema or metametamodel, and so on, in principle ad infinitum. More on this in the chapter on models (see *Framework – models*, p.41).

> The framework is not the architecture, nor are the models the architecture. I know that should be obvious to everyone, but painful experience indicates that it still isn't so for many would-be architects. Time and again we've seen cases where architects have spent many years and many millions of whatever currency in producing a beautiful set of models that are completely useless in practice – and unsurprisingly their clients are not happy about it, or about enterprise architecture in general… Often there's a fair bit of work needed to show that architecture *can* be useful – and the first part of that is to explain what models really do.

> An architecture model is a map, a description of some aspect of the business world; but we need to remember that the map is not the territory, and ultimately the territory is what we need to work on, not the map. As we saw earlier with business-drivers, the *real* end-product of architecture is a dialogue between stakeholders – not a drawing that makes sense to architects but probably to no-one else. Ignore this point at your peril!

A model exists in a context – how we *use* the model. In conventional IT-architecture, a model is essentially about structure and infrastructure, about relationships between 'things' in a physical or virtual sense. In essence, the aim is to *engineer* the information-technology of the enterprise, viewing every aspect of the IT as a component within a vastly complicated machine. But as we saw earlier (see 'A matter of metaphor' in *Purpose – an overview*, p.17), the idea of 'engineering the enterprise' is fraught with problems at the whole-of-enterprise scale. We need to think more in terms of 'organisation as organism', extending the concept of service-oriented architecture, for example, to the whole-of-enterprise scale with a 'viable services' approach. More on how to model this in practice in the chapter in framework integration (see *Framework – integration*, p.42).

The framework described here is what we used in our own architecture practice, but it is of course just one tool amongst many: and do feel free to amend it as you need for your own enterprise.

Application

- What frameworks and other conceptual models do you use to guide your current architecture?
- In what ways have you adapted the underlying metamodels to better describe your enterprise?
- What repositories and toolsets do you use in your enterprise?
- Which frameworks do those toolsets support?

Resources

- Zachman framework: see www.zifa.org
- TOGAF and Zachman: see www.opengroup.org/architecture/togaf8-doc/arch/toc.html (chapter 'ADM and the Zachman Framework')

FRAMEWORK – LAYERS

Summary

For the vertical dimension of the framework, we partition scope in terms of timescale – a set of seven distinct layers or perspectives, from unchanging constants, to items which change moment to moment.

Details

Layers – an overview

If you're familiar with Zachman's framework, its six layers provide us with a starting-point that's essentially as valid in non-IT contexts as in IT-centric ones. The only change we need for whole-of-enterprise architecture is an additional topmost layer.

Strictly speaking, the new layer is a kind of additional dimension, but for practical purposes it's simplest to add it as a row, with a single cell straddling across all the columns, as a common space for 'universals' to which *everything* in the whole enterprise should comply - vision, values, core principles, core policies and the like. (This 'row zero' represents the core architecture-content discovered in the TOGAF-style 'Preliminary Phase' – see *Methodology – preparation*, p.51). All the other rows are split into the respective column cells, as in the original Zachman.

So we end up with a vertical axis like this:

- *Row 0*: '**Universals**' – core constants to which everything should align - identifies the overall region of interest and the key points of connection shared with enterprise partners and other stakeholders
- *Row 1*: '**Scope**' (*Zachman*: 'Planner') – core entities in each category, described in list form, *without* relationships between them - the key 'items of interest' for the enterprise

- *Row 2*: '**Business**' (*Zachman*: 'Owner') – core entities described in more detail, from a business-metrics perspective, including relationships between entities both of the same type ('primitives') and of different types ('composites') - summary form, without attributes for entity-types
- *Row 3*: '**System**' (*Zachman*: 'Designer') – entities expanded out into implementation-*independent* designs - includes descriptive attributes
- *Row 4*: '**Develop**' (*Zachman*: 'Builder') – entities and attributes expanded out into implementation-*dependent* designs, including additional implementations for relationships - for example, cross-reference tables for 'many-to-many' data-relationships
- *Row 5*: '**Implement**' (*Zachman*: 'Sub-contractor' or 'Out of Scope') – implementation of designs into actual software, actual business-processes, work-instructions, hardware, networks etc
- *Row 6*: '**Operations**' (*Zachman*: implied but not described) – individual instances of entities, processes, etc as created, modified, acted on etc in real-time operations

[[see published book for further details]]

Layers – an example

FRAMEWORK – PRIMITIVES

Summary

Below the core-constants, we split the framework horizontally into columns, identifying six distinct major categories of primitives – roughly speaking, what, how, where, who, when and why. In the lower, more implementation-oriented layers of the framework, we also need to split the columns themselves by context into distinct segments or sub-categories – typically related to people, information, things and essential abstracts.

Details

Primitives – an overview

Unlike the Zachman-framework's rows, we really do need to re-think the columns, pretty much from first-principles, in order to make the framework usable for real enterprise-scope architecture. To understand why, we need to look at a little history.

Way back in the 1980s, Zachman defined just three columns for his original framework:

- *What* (data)
- *How* (function)
- *Where* (network)

The trouble is that all this gives us is records (data) of actions (function) that happened or should happen somewhere – which from a design perspective in practice is almost meaningless. Hence quite early on, Zachman added another three columns:

- *Who* (people)
- *When* (business cycle)

- *Why* (business rule)

All well and good – sort of. But as soon as we want to move beyond even fairly simple data-and-function modelling, the limitations soon become apparent. For example, how do we map even basic physical IT-hardware such as servers and routers? They should probably go into the 'What' column – except Zachman says that only holds references to data. Similarly, we can map manual use-cases to the 'Who' column – but there's nowhere to put an automated or IT-based use-case, which we'll need for process re-engineering or service-oriented architecture. Oops…

And there are other, more subtle confusions, too. For example, Zachman correctly asserts that architecture depends on primitives: yet his 'Why' column, he says, consists of 'ends and means'. 'Means' are clearly 'How', not 'Why' – hence 'ends and means' are a composite, not a primitive. As we'll see later, there are other even worse examples of this kind of confusion. So if we use Zachman's framework as-is, we soon find ourselves in a taxonomic black hole – creating real difficulties for design and redesign.

I've seen various attempts to resolve the problems by adding new columns – for 'interface', for example, or 'service'. But in practice, whilst they can seem easier to use for *solutions*-design, at the *architecture* level they just make things worse, a further blurring of the crucial boundary between primitives and composites. At the root-level, the framework *must* be composed of true primitives.

So here's my version for a recommended rework:

- *What*: **assets** of any kind – physical objects, data, links to people, morale, finances, etc
- *How*: **functions**, activities or services, usually to change something – described independently from the agent (machine, software, person etc) that carries out that activity
- *Where*: **locations** – in physical space (geography etc), virtual space (IP nodes, http addresses etc), relational space (social networks etc) and suchlike
- *Who*: **capabilities** and 'responsibilities', often partitioned into roles, as in the 'actor' of a use-case, or a 'swim-lane' on a process-diagram – which may be human, a machine, a software application, etc, and may be either individual or collective (such as the responsibilities of an organisational unit)
- *When*: **events** and relationships between those events – which may be an event or cycle in time, but might equally be physical (a flood, a system-failure), human (a client initiating a business-process), a trigger from a business-rule (a sensor value, a boundary-condition), and so on

- *Why*: **reasons**, decisions, constraints and other tests which trigger or validate the condition for the 'reason' – as in strategy, policy, business-requirements, business-rules and so on.

The end-result of this rework is that down at the row-5 / Implement level, it should be possible to describe *everything* in terms of a single sentence-structure for work-instructions. This maps to the revised columns as a distinct set of primitives, and as a composite straddling across every column:

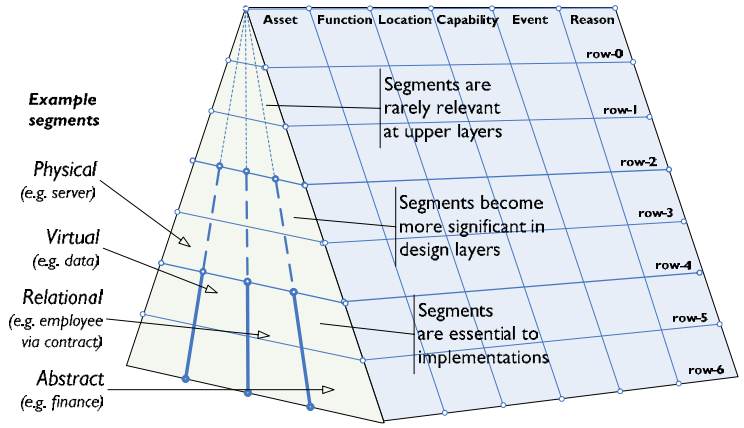
"with <*asset*> do <*function*> at <*location*> using <*capability*> on <*event*> because <*reason*>"

At the row-6/Operations level, the sentence might come out in a slightly different order, and usually in the past tense, but it should still be essentially the same:

"<*function*> was done with <*asset*> by <*capability*> at <*location*> on <*event*> because <*reason*>"

And although it's a composite, we can still identify the individual component-primitives that make up that composite. Which makes architectural redesign possible, and which anchors the trails of relationships between items and between layers that we need in order to resolve *business*-concerns such as strategic analysis, failure-impact analysis and resolution of complex 'pain-points'.

The other point is that in effect there's an entire *dimension* missing from the Zachman framework, to split each column into distinct and necessary *segments*. As in the examples above, we need to be able to distinguish between physical items, virtual items, people-relationships-as-items; or functions done by machines, by software, by people; and so on.



Rows, columns and segments

We'll look at this in more depth as we explore each of the columns in turn.

[[see published book for further details]]

Assets – 'what?'

Functions – 'how?'

Locations – 'where?'

Capabilities – 'who?'

Events – 'when?'

Reasons – 'why?'

Synopsis

FRAMEWORK – COMPOSITES

Summary

Architectural primitives are often of little or no value on their own: to be usable, they need to be linked together into composite entities which represent some real-world entity or pattern. Composites will often be layered into more complex entities, moving towards a description that is complete enough to use in practice.

Details

[[see published book for further details]]

Composites – an overview

Root-composites

Some example root-composites

Complex composites

Some example complex-composites

Composites and 'completeness'

FRAMEWORK – MODELS

Summary

Models are often the most visible part of architecture work. Each model-type represents a different way to describe a specific business context, in terms of architectural entities and relationships between those entities. At the whole-of-enterprise scope, there are significant issues with access-control, model versioning and model-dynamics that need to be addressed.

Details

[[see published book for further details]]

Models – an overview

Models, views and viewpoints

Models and their entities

Models and relationships

Models and metadata

Model versioning and model dynamics

FRAMEWORK – INTEGRATION

Summary

At root, the framework needs to be able to describe, visualise and model every aspect of the enterprise – whatever form the enterprise may take, and whatever way it may change. This requires a great deal of flexibility, not just in the framework itself, but in the way we perceive the enterprise and the architecture process.

Details

[[see published book for further details]]

Integration – an overview

Integration – organisation as organism

Integration – viable services

Integration – boundary effects

METHODOLOGY – AN OVERVIEW

Summary

A formal methodology specifies the structure and sequence – the how, where and when – for architecture activities. The methodology described here is derived from the Architecture Design Method (ADM) of the Open Group framework (TOGAF), adapted to suit the broader scope and complexities of whole-of-enterprise architecture.

Details

Methodology and governance

Purpose identifies the 'why' of the architecture-cycle, governance circumscribes the 'who', and the framework describes much of the 'what', the means to derive meaning from what is discovered. Next we need methodology, to specify the 'how', 'when' and 'where' – the structure and sequence for the activities, the actual *process* of architecture.

The methodology is where we start to put all the ideas into practice. Up till now it's all been somewhat theoretical, abstract; but here we need to *do* something – and something that has practical *use*.

There are two radically different approaches we can take to this. In the classic 'hands-on' approach, which we'll explore in this and the next few chapters, architecture drives design: there'll be a strong emphasis on development of 'blueprints', 'roadmaps for change', and standard reference-models, and an insistence on compliance to those models. This style is typical, and usually appropriate, in large organisations and in the earlier stages or architecture development. The catch is that unless we're careful, it can become cumbersome and bureaucratic, ensuring compliance at the cost of agility. To resolve this, the other approach, which we'll explore later in *Methodology – hands-off architecture* (see p.63) goes to the opposite extreme, allowing design and consistency to emerge from the natural complexity of everyday architectural choices.

Whichever way we do it, we need strong links to governance, to ensure that what we do *is* of practical use to the enterprise. The governance processes are there to ensure that architecture does indeed begin with a business purpose; those processes ensure appropriate actions by the appropriate people at each stage in creating and implementing the architecture.

Methodology implements governance;
governance maintains accountability in the methodology

So whilst we might describe the activities in terms of a methodology, we need to embed governance *within* that methodology. The TOGAF ADM methodology does this for IT architecture, for example, in calling for stakeholder reviews at various points in the ADM cycle.

But here we hit a problem of scale. The classic 'big bang' approach to enterprise architecture requires us to develop 'as-is' and 'to-be' architecture descriptions for the entire scope – which could literally take years, just for the IT-architecture alone, and most probably be well out of date long before the process is completed. But for anything smaller, TOGAF's requirement for *dozens* of stakeholder-reviews is clearly going to be overkill: if we call a stakeholder-review every week, it won't be long before no-one bothers to turn up – which in effect means no governance. So we need an approach to governance that will give us the right balance for architecture-cycles at every scale, from quick assessment-projects to mid-size change-programmes and portfolio management, up to full-scale business transformation.

The same applies to the methodology itself: it too needs to be usable *and consistent* at every scale. We need something that is scalable, yet in essence uses the same steps and sequence, whether the business purpose requires activities that will take half a day or half a year. The classic IT-architecture methodologies, such as FEAF and TOGAF, only describe the full-scale 'big-bang', in a strict 'Waterfall' style: it's clear we'll to rethink this approach if we're going to have a methodology that we *can* use at every scale.

> It must be dozens of times now that would-be clients have asked us to develop an architecture "in accordance with the Zachman methodology". They often get quite upset when we explain that we can't do so, for the simple reason that it doesn't exist. Zachman himself has been promising for years to provide one, but to date he still hasn't published anything: the nearest that's available is in his architecture seminars.
>
> But in the absence of an explicit methodology, people often invent something, *anything*, that seems to fit the Zachman framework. So given Zachman's exhortation that ultimately we need to assess everything in "excruciating detail", the putative architects plod their way through every cell of the original framework, twice – once for 'as-is', and once for 'to-be'. Which is not only insanely time-consuming, but also, as we've seen, too misleading and incomplete even for IT-

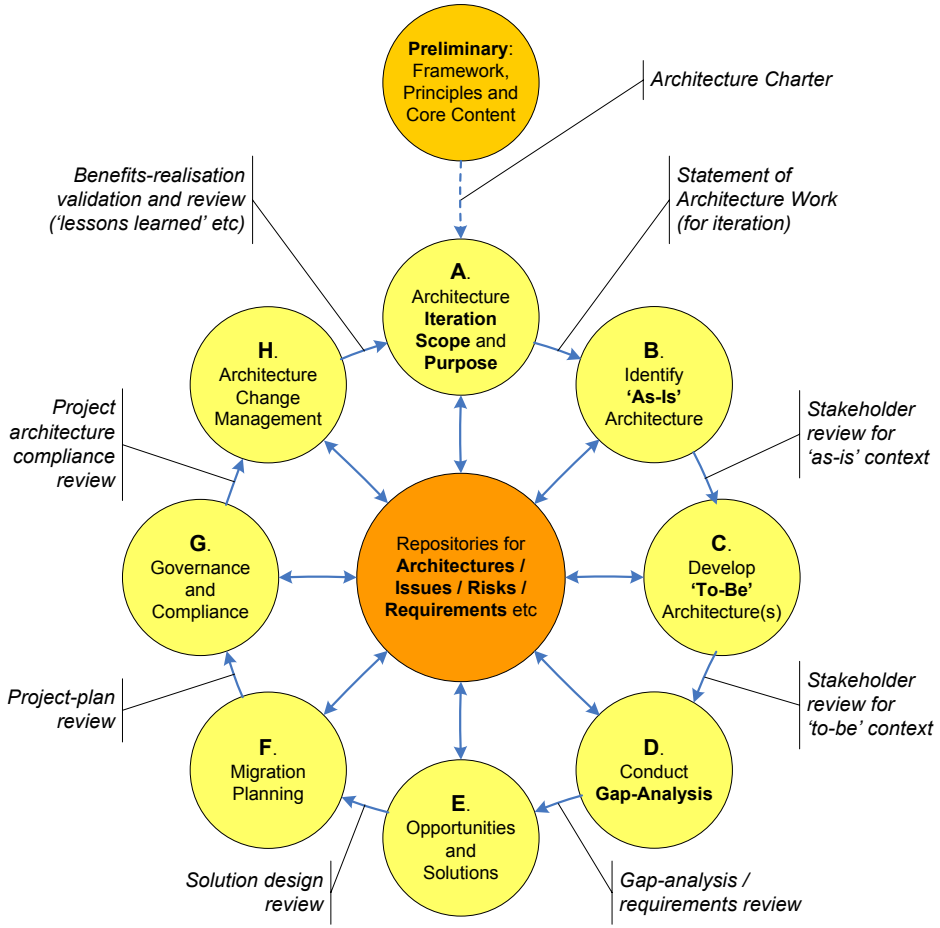
architecture, let alone true whole-of-enterprise architecture. Worse, there's no means to link in any kind of systematic governance at all – a guaranteed recipe for problems. *Not* recommended…

Of the current crop of methodologies, TOGAF version 8.1 'Enterprise Edition' is probably our best choice. The original ADM is perhaps more of a hindrance than a help for whole-of-enterprise architecture, because it's set up only for 'big-bang' development, and so rigidly IT-centric that *everything* not-IT is dismissed as 'Business Architecture'. But we can make it usable for any scale and scope of architecture-work by tweaking the definitions and the sequence of some of the activities in some of the ADM's Phases – particularly in Phases A-D – and adding more systematic links to governance.

The architecture cycle

The architecture cycle in the TOGAF ADM in effect follows the same project-lifecycle as we're using here – Purpose, People, Preparation, Process and Performance. This is perhaps not obvious because there are *two* interleaving cycles: one for the assessment phases – the architecture proper – and one for solution-designs arising from the assessments. At this level, the main difference between the ADM and what we're doing here is that we're not restricted to constructing an IT-architecture: we can use the same methodology and Phase sequence for architecture at *any* scale, and for any framework scope.

This modified TOGAF ADM cycle also maps well with PRINCE2 and similar programme or project management methodologies and governance. For example, unlike the original ADM, we now have just one explicit stakeholder-review at the end of each of the Phases A to D. The key governance documents or 'products' also mark the boundaries between the architecture-cycle Phases.



Governance-artefacts define architecture-cycle's phase-boundaries

Before we can do any architecture development, we need to set up the architecture capability itself. To do this, we 'boot' the architecture via a high-level version of the *same* architecture-cycle. In TOGAF, this is described as the 'Preliminary Phase'.

- **Phase P**: Preliminaries – establish (or review) the architecture capability, the purpose, governance, framework, methodology and completion, and define a big-picture view of what the overall aims to achieve for the entire enterprise

During the preliminary phase we define the key documents such as the Architecture Charter, and also set up the repositories and registers that are at the core of the architecture cycle: requirements, issues and risks, glossary and thesaurus, architecture-dispensations, governance-records and the architecture repository itself.

Once this is done, and the governance and the like formally approved, we're now ready to fill the detail of the architecture and apply it in practice. A key point here is that unlike the assumptions in IT-centric architectures such as FEAF or TOGAF, we *know* that what we're dealing with at a whole-of-enterprise level is too large to tackle in one go. So we start from the clear knowledge that we're going to do this iteratively, with each step constrained within the relatively narrow scope of a single business-issue or change-project. This approach works because we're using a framework that has a true enterprise-wide coverage. It also makes better business sense, because we gain some immediate return from the architecture work, and the value increases steadily as each project leverages the knowledge and lessons learned from previous architecture cycles.

In the TOGAF specification, the scope and purpose are predefined: it's always IT-centric architecture, with a strong emphasis on detail-level technology. But here we may be dealing with *any* scope, *any* business-issue. So if you're familiar with TOGAF, note that the first four Phases here each have a subtly different emphasis from what you may be used to:

- **Phase A**: Establish Iteration Scope – identify the core business-issue(s) to be addressed, and scope (in terms of framework layers, columns and segments) to be covered in the analysis
- **Phase B**: Assess Current Context – establish the current architectural description for the scope and business-issue identified in Phase A
- **Phase C**: Assess Future Context – establish the required future-architecture for the scope and business-issue identified in Phase A
- **Phase D**: Derive Change Requirements – establish the gaps between the current architecture (from Phase B) and desired future architecture (from Phase C), and the resultant change-requirements and constraints in relation to the scope and business-issue (from Phase A)

At the end of Phase D, we should have a set of business requirements, which provide the *purpose* for subsequent solution-designs. Although the architecture-assessment cycle doesn't complete until Phase H, the lessons-learned stage, the high-level architects drop back to a support-role, handing over to the solution-architects. This second part of the cycle is much closer to the original TOGAF ADM: the only key difference is that, again, it can cover more than just an IT-centric scope:

- **Phase E**: Design Solutions – work with solution-designers to assess options and trade-offs between requirements and constraints (from Phase D) to identify high-level solution-designs for the required changes
- **Phase F**: Plan Migration – work with governance, portfolio and change management teams to develop transformation blueprints, change-programmes and individual implementation-projects
- **Phase G**: Guide Implementation – work with programme and project managers to assist in resolving trade-offs between architecture and implementation

When all projects arising from the assessment are complete, the high-level architects and solution-architects alike need to carry out a 'lessons learned' exercise, not only to identify any architectural concerns that might trigger new architecture-cycles, but also to review core definitions for the architecture itself: standards such as key components in the framework, the glossary and thesaurus, and even the underlying Architecture Charter.

- **Phase H**: Review Lessons-Learned – assess all issues arising from the architecture cycle, and identify (and, where appropriate, implement) any required changes to architecture standards and processes

The architecture is never 'complete': but it never pretends to be, either. Instead, it grows and changes with each iteration, creating a richer and more valuable view of the enterprise as a whole.

Dynamic architecture

There's an important complication here in that the architecture is necessarily *dynamic*. Architectural assessment may be quick, but implementation often is not, and the architecture may well have moved on by the time implementation catches up. So both the architecture cycle and the architecture itself need to be designed to work with this stark fact.

The problem is that most existing approaches describe an architecture in terms of a 'current-state' and a desired 'future-state'. We can just about get away with this for a simple narrow-scope IT-architecture; but the reality is that *nothing* is static, there *is* no 'state' – especially at the 'city-plan' level implied by true enterprise wide architecture. Everything depends on everything else, and everything is changing, at different speeds and in different ways. Complex, to say the least.

To support dynamics properly, we need a sophisticated system of versioning and prioritisation, with dependencies mapped to different versions as required. The catch is that whilst there's no doubt that we need a sophisticated toolset to manage all this complexity, none of the current crop of toolsets come anywhere close to what we actually need – see *Basics – artefacts and toolsets*, p.15. Some don't really handle versioning at all, whilst the only toolset with a halfway-decent versioning system has no scripting language, so we can't do much with the versioning anyway. And we need consistent versioning and mapping across *all* of the architecture's repositories – not just modelling, but requirements, issues, compliance and everything else – so it doesn't actually help us if the requirements versioning is excellent and the model versioning all but non-existent, as is the case with one well-known toolset.

So until the toolset vendors wake up to the complexities of enterprise level architecture, we're still on our own to some extent in managing architecture dynamics. There *are* workarounds in every case, though inevitably all are somewhat labour-intensive and somewhat fragile; they also depend on the toolset in use, so I can't give specific advice here. The first and most important requirement, though, is simply to respect that complexity, design for it, work *with* it – and not hide from it in misleading terms such as 'future state'!

Application

- What formal methodology – if any – do you use for your current architecture?
- What are the boundaries of that methodology – for example, does its scope address only IT-systems or IT-services?
- How do you address architectural concerns beyond the nominal scope of the methodology?
- How do you address the dynamics of architecture – versions, interactions, lifecycles and so on – within the methodology?
- If you use an architecture toolset, in what ways does the toolset support or constrain the methodology?
- What processes and products are used for governance of that methodology?

In answering those questions, consider what you'd need to change in order to make the methodology more suitable for the broader scope and more complex dynamics of whole-of-enterprise architecture.

Resources

- TOGAF ADM: see www.opengroup.org/architecture/togaf8-doc/arch/toc.html (chapter 'Introduction to the ADM')

METHODOLOGY – PREPARATION

Summary

Before we conduct any business-critical architecture, we need to decide on and verify the details of the governance, framework and methodology we are to use. We also need to review this overlighting structure on a regular basis, to ensure that it aligns with the changing needs of the enterprise.

Details

Preparation – an overview

In terms of the methodology, preparation for architecture consists of a single phase, described as 'Phase P'. In essence, this is a standard architecture cycle, but applied to the architecture itself – hence there's a strong emphasis on issues such as standards, principles and governance. It's also during this phase that we secure or renew essentials such as funding and authority from the executive to do the work.

Phase P – preliminaries

This phase is all about defining "how we do architecture". We identify the overall scope of enterprise-architecture, the authorisation from the executive to conduct enterprise-architecture, the architecture team and their roles, responsibilities and function within the organisation. It's also essential in this phase to outline the overall governance, frameworks and methodologies to be used for architecture development and architecture services.

If you're familiar with TOGAF 8, most of this should be well-trodden territory. The main difference here is that we've merged much of TOGAF's 'Phase A' into this part of the process.

The reason is that TOGAF 8 not only has a fixed scope, but also tends to do things in a big way: although in principle it's an iterative process, a typical TOGAF ADM cycle will take many months, if not years. The Preliminaries there are all the governance things – Principles, Charter and so on – whilst Phase A sets up the fine details of the scope within the fixed TOGAF framework. In this revised ADM, though, we could be working with any scope, but more often small than large:

a complete architecture iteration might take as little as a few hours in some cases, and still deliver business-usable results. So whilst the detail-scope parts of Phase A remain where they are, we move the definition of the overall framework, and operational setup, to become part of the preliminaries here, so that we have everything that we'll need already in place before we start any active architecture work.

Key products include the Architecture Charter, Architecture Principles and core high-level content for the architecture framework.

This phase is independent of the main architecture cycle. Since it provides oversight to the main cycle, we need to do it at least once before any architecture work takes place, but we also need to revisit it at regular intervals – for example, as a formal annual review.

Objectives, inputs and outputs

The objectives of the Preliminary Phase are:

- ensure that everyone who will be involved in, or benefit from, this approach is committed to the success of the architectural process;
- ensure that this evolution of architecture development has proper recognition and endorsement from corporate management, and the support and commitment of the necessary line management;
- validate the business principles, business goals, and strategic business drivers of the organisation;
- define architecture principles that will inform the constraints on any architecture work;
- define the organisation's 'architecture footprint' – the people responsible for doing the architecture work, where they're located, their responsibilities and so on;
- define the scope and assumptions (particularly if there's a 'federated architecture' to be shared with others);
- define the overall framework and working details of the methodologies that are to be used to develop architectures within the enterprise;
- evaluate and confirm the selection of architecture tools, repositories, and repository management processes that you will use to capture, publish, and maintain architecture artefacts;
- define the scope of the architecture effort, and identify and prioritise its components;
- identify the relevant stakeholders, and their concerns and objectives;
- identify the key business requirements for the current architecture effort, and any applicable constraints;

- secure formal approval to do the work.

The inputs (►) and outputs (◄) created in the Preliminary Phase are:

► Request for Architecture Work
◄ Governance documents such as Architecture Charter, Architecture Governance, Architecture Principles, Architecture Standards
◄ Architecture-models repository ('Enterprise Continuum')
◄ Requirements repository
◄ Glossary and thesaurus
◄ Issues/ risks registers
◄ Architecture-dispensations register
◄ Phase-completion report – stakeholder review of preliminaries

If a version of an output document already exists, it will be an input to the Phase, and may be amended during the Phase.

Steps

Typical steps include the following:

Step P1 – Establish the enterprise-architecture capability

Follow your own organisation's procedures to secure enterprise-wide recognition of the architecture capability; endorsement by corporate management to do the work, and crossing organisational boundaries where necessary; appropriate funding to establish and maintain the capability; and the support and commitment of any line management whose work would be affected by the architecture.

Include by reference the enterprise's governance framework, explaining how the architecture capability relates to that framework (for example, as a support function for an enterprise-wide programme-management office). Document this in the Architecture Governance document.

Under those governance rules, prepare a Request for Architecture Work that covers the scope and context for the remainder of the work for this Preliminary Phase.

If they don't already exist, create the required shared-information repositories:

- Architecture-models repository
- Requirements repository
- Issues register
- Risks / opportunities register
- Architecture-dispensations register
- Glossary and Thesaurus

Record the results of this step in a preliminary draft (or review-draft) of the Architecture Charter document. (For the Preliminary Phase, the Architecture Charter acts as the equivalent of the Statement of Architecture Work – in other words a formal record of architectural decisions, actions and results.)

Step P2 – Identify Architecture Principles

Review the principles under which the enterprise architecture will be developed. Architecture principles are usually either based on or extensions of the core principles of the enterprise, including the organisation's Vision and Values. Ensure that any existing definitions are still current and valid, and clarify any areas of ambiguity. Otherwise, go back to the responsible governance-body, and work with them to define these essential items from scratch and secure their endorsement by corporate management.

Record the agreed principles in the Architecture Principles document.

Amend the Architecture Charter document as appropriate, to reference the applicable principles and so on.

Step P3 – Identify applicable business policy, legislation and regulations

Review any policies and regulations, whether internal or external, which the enterprise-architecture capability must reflect and implement. Typical core policies include security policy, OH&S policy, privacy policy, environment policy, quality-system etc. Ensure that any existing definitions are still current and valid, and clarify any areas of ambiguity. Otherwise, go back to the respective bodies within the business, and work with them to resolve any issues and secure endorsement by corporate management.

Record the results of this step in the architecture-repository as entries in the 'Universals' layer of the framework

Amend the Architecture Charter document as appropriate, to reference the applicable policies and suchlike.

Step P4 – Identify applicable Standards

Repeat the above for any applicable Standards, whether internal or external, which the enterprise architecture capability must reflect and implement. If necessary, go back to the respective bodies within the business, and work with them to resolve any issues and secure endorsement by corporate management.

> Note that the framework and the structures and metamodels for each of the Enterprise Continuum repositories are also in effect Standards for your enterprise architecture, and need to be identified and managed as such here.

Record or reference the Standards in the Architecture Standards document. If appropriate, also create respective architecture-repository entries in the 'Universals' layer of the framework.

Amend the Architecture Charter document as appropriate, to reference the applicable Standards etc.

Step P5 – Identify core business-goals and business-drivers

Apply the same review-process with any core business goals and strategic drivers of the organisation which are deemed to apply for the time-periods covered by the architecture.

These should usually have already been defined elsewhere within the enterprise. If so, ensure that the definitions are still current and valid, and clarify any areas of ambiguity. Otherwise, go back to business groups responsible for definition of such goals and drivers and work with them to define these essential items from scratch and secure their endorsement by corporate management.

Record the results of this step in the architecture-repository as entries in the 'Universals' layer of the framework.

Step P6 – Identify enterprise-architecture scope

Define what is and isn't within the scope of the intended enterprise architecture capability. In particular, define:

- breadth of coverage of the enterprise
- applicable time horizons to be supported for 'future-context' assessments
- any architectural assets to be leveraged, or considered for use, from the organisation's Enterprise Continuum:
  - assets created in previous architecture efforts within the enterprise

- assets available elsewhere in the industry (frameworks, systems models, vertical industry models, etc.)

From these, determine which overall architecture domains should be developed, to what level of detail, and which architecture views should be built. Ensure that the Architecture Repository includes metamodels – see *Framework – an overview*, p.28 – sufficient to cover all aspects and views for this scope.

Amend the Architecture Charter document as appropriate to describe or reference the applicable scope.

Step P7 – Identify constraints

Identify the operational constraints that must be addressed, including enterprise-wide constraints and any project-specific constraints (time, schedule, resources, etc.). The enterprise-wide constraints may be informed by the business and architecture principles developed in the steps above.

Amend the Architecture Charter document as appropriate, to reference the applicable constraints.

Step P8 – Identify stakeholders and concerns, business requirements, and overall architecture Vision

Identify key stakeholders and their concerns and objectives; define the key business requirements to be addressed in this architecture effort; and articulate an Architecture Vision that will address the requirements, within the defined scope and constraints, and conforming with the business and architecture principles.

Record the concerns and requirements, and their respective stakeholders, as entries in the Requirements Repository. (If requirements are maintained within the architecture repository, record the requirements as entries in the 'Why' [reasons / decisions] column of the framework.)

Amend the Architecture Charter document as appropriate, to reference the applicable stakeholders and the overall Architecture Vision for the current development.

Step P9 – Identify content for high-level models

Use the core-principles, scope, requirements, constraints and the like from the above steps to suggest candidate items for each cell in the 'Planner' (row-1) layer of the framework:

- *what*: key data-items, key physical 'things' etc
- *how*: key processes, functions etc

- *where*: key locations, physical, virtual etc
- *who*: key capabilities, actors/agents, organisations etc
- *when*: key events, cycles etc
- *why*: key continuing aims, strategies etc

Record the results in the respective sections of the architecture-repository. Depending on the capabilities of the toolset, these may be in either list- or graphic-form, but it must be possible to link to these from other layers of the stored framework.

Note that in the first pass through the Preliminary Phase, you may only have minimal content to store, but you'll add further content during architecture cycles and subsequent Preliminary Phase reviews.

Step PX – Secure approval for Architecture Charter, governance, etc

Estimate the resources you'll need to operate the architecture capability. If it's a new capability – for example, if this is the first time you've done a Preliminary Phase – develop a roadmap and schedule for the proposed development; otherwise identify the existing resources and any changes required for the period under review

Amend the Architecture Charter to include these items, usually by reference.

Secure formal approval of the Architecture Charter under the appropriate governance procedures. Once you've done this, the Architecture Charter, together with the applicable documents for Principles, Governance, Standards and so on, will act as the formal authority to conduct architecture work as subsequent iterations through the Phase A to Phase H cycle.

Application

- What methodology do you use at present for architecture?
- What processes does it provide for preparation for architecture?
- What formal process – if any – do you use for managing, reviewing and updating the methodology itself?
- In what ways – if any – is the current architecture linked to the high-level 'universals' of the enterprise?

- What is the current scope and authority for 'enterprise architecture'? Under whose governance? If it does not have a true enterprise-wide scope and authority, what would need to change in order to attain these?

Resources

- TOGAF and ADM preliminary phases: see www.opengroup.org/architecture/togaf8-doc/arch/toc.html (chapters 'Preliminary Phase: Framework and Principles' and 'Phase A: Architecture Vision')

METHODOLOGY – ASSESSMENT

Summary

The first half of the architecture-cycle is focussed on structure and high-level design: 'architecture proper'. Starting with a business need, we explore impacts and options, and derive requirements for change – adding to our knowledge of the enterprise structure as we do so.

Details

Assessment phases – an overview

It's in the assessment phases – Phases A to D of the architecture-cycle – that we create most of what other people would see as 'architecture': the models and design-requirements and suchlike. There's also a strong emphasis here on the link between methodology and the content and maintenance of the architecture framework.

If you're familiar with TOGAF, you'll need to do a slight readjustment in your thinking here. In principle it's much the same, but we do things in a different order, and often much more quickly than in the original ADM. We've also done some of the preliminary work already, so Phase A here has a slightly different emphasis and function.

The big change, of course, is that we don't assume TOGAF's fixed scope in each Phase, as 'Business Architecture' (original Phase B), 'Information-Systems Architecture' (original Phases C1 and C2) and 'Technology Architecture' (original Phase D). Instead, we allow an architecture cycle to have *any* scope – and it's that that we sort out in the revised Phase A.

In the original ADM, we would do a separate as-is, to-be and gap-analysis in each Phase – which made sense there, because the predefined scope for each of their Phases would typically involve different stakeholders. But here we'll typically expect to engage the same stakeholders all the way through, so it makes more sense to split the work on the time-horizon boundaries instead.

If you want to use this variant to do 'classic' TOGAF-style IT-architecture, think of that as three distinct, successive passes through the revised assessment-Phases A to D. One pass has a 'Business Architecture' scope (typically higher-level Zachman); one has an 'Information-Systems Architecture' scope (typically mid-level Zachman, and possibly split in

two for a separate Applications scope around *capability* and *function*, and a Data scope around *asset* » virtual); and the last has a 'Technology Architecture' scope (mostly low-level Zachman around *asset* and *location*). You'd then bring all of these together in an additional Phase D that merges and cross-compares all the different gap-analyses, leading on to the solution-phases in the usual way with a single Phase E. The original ADM does allow you to go back and forth in this way anyway: it's just that here we can be a bit more explicit about it, and do it under more explicit governance.

Not recommended to do it all in one go, though – it's all too big to be workable in practice, as many enterprise-architecture teams have found to their cost. It's much simpler, and much more reliable, to do it in smaller, more manageable chunks, each directly linked to an identifiable business need. But if the business really do insist that they want the classic 'big bang', there's nothing here that would stop you from doing it – go right ahead, and have fun as best you can, perhaps?

Since architectural design is the centre of attention here, rather than the details of implementation, the focus of governance is more on the architecture itself than programme-management – though the latter should at least be informed, and preferably engaged, in every step of the process.

[[see published book for further details]]

Phase A – establish iteration scope

Phase B – assess current context

Phase C – assess future context

Phase D – derive change-requirements

METHODOLOGY – SOLUTIONS

Summary

The second half of the architecture-cycle is focussed on detail-level design, implementation and deployment, and, at the end, lessons-learned from the iteration. Although domain-architects may be intensely involved in many of these processes, the enterprise-level architects should pull back to more of a support-role, maintaining a proactive 'watching brief' for any enterprise-wide architectural concerns.

Details

Solutions phases – an overview

In the solution phases – Phases E to H of the architecture-cycle – the emphasis shifts away from enterprise-level architecture, and more towards a background support for the detail needed to design, implement and deploy the changes implied by the requirements identified in the gap-analysis. The focus of governance here moves to project- and programme-management, with enterprise architecture called upon mainly to provide architectural guidance and arbitration between projects,. and to maintain high-level consistency as the architecture changes dynamically over time.

If you're familiar with TOGAF, this too will be well-trodden ground – and in effect is much the same as in the original ADM, too, because most of it isn't architecture's business anyway. Whilst the domain-architects will be busy, of course, the main role for *enterprise*-architecture in these Phases is a watching-brief, providing architectural support and 'big-picture' overview, but otherwise keeping out of the way unless asked. Architecture is not design: and designers understandably do not like it when architects meddle in their work without good reason!

As before, the main difference is one of scope. whilst TOGAF expects all solutions to be centred on IT, we shouldn't make any such expectation. In fact some, if not many, of the solutions here may involve no IT at all – which could well be a challenge at first if you've come from a strict IT-centric background. If you keep the focus on overall *effectiveness* – 'efficient *on purpose*' – you should be able to stay safely on track to support whatever's required.

The real complexity here is in the trade-offs between architectural 'purity' and real-world constraints. The most valuable skill of the enterprise-architect is the ability to find a balance between what is desirable and what is available and achievable, to derive solutions that are truly effective – efficient, reliable, elegant, appropriate, integrated – across the *whole* of the enterprise. Tact and diplomacy are essential character-traits here: consistency is important, but trying to enforce compliance via the 'architecture police' is *not* the way to do it…

For the same reasons, perhaps the greatest danger here is one of ego. Too often we've seen architects not only destroy their architecture, but damage their own careers and the credibility of architecture as a whole, in the hubris of trying to play 'the Creator of all' and suchlike. It's essential to remember that at the end, the architect is just one member of a much larger team tasked with creating change in the enterprise: it *is* important to stand up for our truth, but the ultimate authority here belongs to programme-governance, not to architecture.

[[see published book for further details]]

Phase E – design solutions

Phase F – plan migration

Phase G – guide implementation

Phase H – review lessons-learned

METHODOLOGY – HANDS-OFF ARCHITECTURE

Summary

In complex contexts with high rates of change, a more mature enterprise architecture capability can take a hands-off approach, allowing some aspects of the architecture to emerge naturally from the complexity. Note, though, that use of such a strategy in an inappropriate context or with immature architecture could result in chaos, with expensive results all round: it should not be attempted unless the risks and trade-offs are fully understood and addressed.

Details

[[see published book for further details]]

COMPLETION – AN OVERVIEW

Summary

Architecture is a continuous process, and needs nurturing to grow into ever more useful forms. We do this through communication and feedback, through discussion and dialogue with our business community – and through measuring the value of what we do.

Details

[[see published book for further details]]

COMPLETION – ARCHITECTURE ARTEFACTS

Summary

The most visible end-product of enterprise-architecture is the set of documents and other artefacts created during the various stages of the methodology and its governance. Most of these artefacts should be created, maintained and managed in a controlled, disciplined manner, to provide a stable reference for future use and reuse. This section describes the main types of artefacts or 'products', their contents, purpose and stakeholders.

Details

[[see published book for further details]]

Governance products

[[examples include:]]

- Architecture Charter
- Architecture Governance
- Architecture Principles
- Architecture Standards
- Request for Architecture Work
- Statement of Architecture Work
- Solution Design Document
- Project-plan or Migration-plan
- Architecture Compliance Statement
- Architecture Description Statement

- Architecture Position Statement
- Architecture Dispensation Statement
- Phase completion reports – stakeholder reviews

Repositories

[[examples include:]]

- Architecture-models repository
- Requirements repository
- Issues register and risks register
- Dispensations register
- Glossary and Thesaurus

COMPLETION – CLOSING THE LOOP

Summary

No matter how good the architecture development, it will only have real value for the enterprise if it is applied and put to practical use. The key requirement for this integration is some form of feedback and engagement of stakeholders, not only in the initial development of architecture, but also in its dissemination, review and re-use.

Details

[[see published book for further details]]

Linking everything together

Publish or perish?

Garnering feedback

Setting up for the next cycle

GLOSSARY

This summarises some of the terms and acronyms we've come across in the book.

[[see published book for further details]]