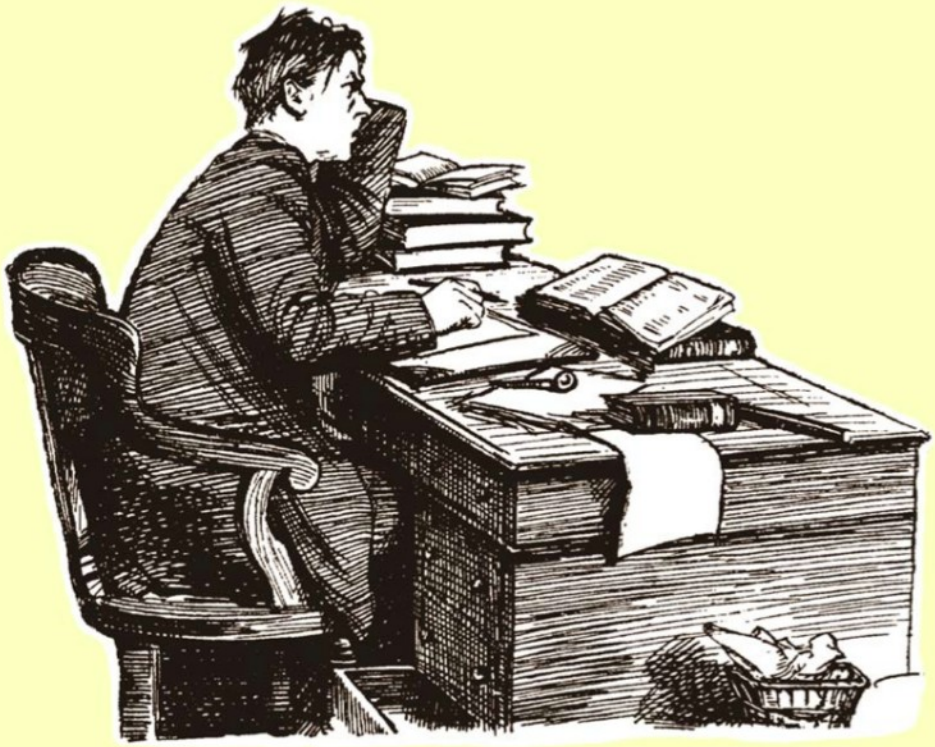
Everyday enterprise architecture

Sensemaking, strategy, structures and solutions



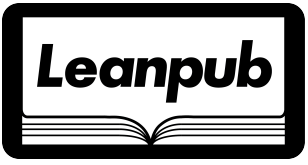
Tom Graves

Tetradian Enterprise Architecture Series

Everyday enterprise-architecture

Sensemaking, strategy, structures and solutions

Last published on 2012-03-11



This is a Leanpub book, for sale at:

http://leanpub.com/tb-everydayea

Leanpub helps authors to self-publish in-progress ebooks. We call this idea Lean Publishing. To learn more about Lean Publishing, go to: http://leanpub.com/manifesto

To learn more about Leanpub, go to: http://leanpub.com

Contents

Everyday enterprise architecture
Sensemaking, strategy, structures and solutions

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 April 2010

ISBN 978-1-906681-24-1 (paperback)
ISBN 978-1-906681-25-8 (e-book)

Acknowledgements

Amongst others, the following people kindly provided comments, advice, suggestions and feedback on various ideas expressed in this book: Sally Bean (GB), Shawn Callahan (AU), Pat Ferdinandi (US), Paul Jansen (BE), Anders Ostergaard Jensen (AU), John Polgreen (US), Chris

Potts (GB), Kevin Smith (GB), Michael Smith (MX), Richard Veryard (GB).

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.

Book contents

The full chapter-list of the book is summarised below; chapters included in this sample-version are shown in *italics*.

Introduction

Enterprise-architecture for a real-time world

What exactly do we *do* every day in enterprise-architecture? What value does it deliver to the business? How do we develop our skills and experience, our judgement and awareness, so as to keep on enhancing the value that we deliver? And how do we do it *fast*, to respond to the real-time pressures of an always-on business world?

Many books on enterprise-architecture place an emphasis on frameworks, models or methods – their overall *theory* of architecture, without much description of what actually happens in day-to-day practice. The reason for that gap is simple: if we start from theory, it's hard to show much more than guidelines and principles without getting lost in irrelevant detail, because every architecture context is different.

So this book takes almost the opposite approach. We concentrate on the everyday *activities* that underpin each of the architecture disciplines – particularly the core processes such as sensemaking and design-thinking.

We explore how and why and when the various items of 'theory-stuff' come into the picture – all those methods, frameworks, models, metamodels and other information-sources.

And we show how to do all of that in a real architecture-project that must deliver real business-value in just two working weeks – not the two years or more required by some other approaches to enterprise-architecture. So yes, *real* enterprise-architecture, in real-time, that really *does* make business sense.

Use the architecture itself to explain how to do enterprise architecture.

Illustrate it by tackling a real enterprise-level business problem.

Exactly ten days in which to do it.

Starting now.

Go.

Interested? Read on…

Who should read this book?

The principles and practice described here apply to every type of enterprise – for-profit, not-for-profit, government, whatever – and at every scale, from a global corporation to the local bowling-club and beyond. So in principle, and in practice too, it should be relevant to just about everyone.

This book should be especially useful for enterprise- and business-architects, but also for executives, strategists, strategic analysts and any others who are tasked with understanding the enterprise as a whole.

This is part of a larger series of books on new developments in enterprise-architecture. Enterprise-architectures provide a 'big-picture' overview for other architecture disciplines: so this would also be useful for process architects, security architects, solution architects, software architects and the like.

What's in this book?

The aim of this book is to show what *actually* needs to happen in enterprise-architecture practice – not just its outcomes, but the activities from which those outcomes arise. As part of this, the book introduces a new technique called 'context-space mapping', which provides a structured method for sensemaking across the entire context of an enterprise. There's also a strong emphasis here on what building-architects describe as 'meta-thinking' – the reflective 'thinking about thinking' through which the quality of personal practice is developed.

The book and its content are built around a real two-week project explicitly undertaken to illustrate all of these themes. Each of the ten main chapters in the book describes the respective day's activities, and includes and expands on the actual project-diary entries for that day, which are shown as follows:

> ***Diary***
> *Diary-entries have their own distinct formatting to separate them from the main text*

There are also various comments, anecdotes and examples – again drawn from real business practice – which are shown as follows:

> A story, anecdote or aside provides a real-world example of the point that's being discussed in the main text.

Most books on enterprise-architecture and the like will include many illustrative models and diagrams, and this too follows that tradition. What's different here is that many of these diagrams are adapted straight from the sketch-pad or whiteboard, to emphasise that this book is all about what happens in real-world practice.

There are actually two projects running in parallel during the two-week period described in this book:

- how to use architecture ideas and activities to describe what actually happens in a real enterprise-architecture project, and the business-reasons and business-value for each of those activities

…and, selected during the early stages of that main project, to illustrate each of the respective principles and practices:

- using architecture to address a real enterprise-level business-problem that was a serious and urgent concern to one of our clients

The architecture activities for this second project are described in a separate section in the later part of each day's chapter.

Each of those chapters ends with another section that provides suggestions for how to apply the same principles in your own architecture work.

There are also a couple of extra chapters after the overall projects. The first of these describes the structures of the information-repositories needed for enterprise-architecture, and summarises the respective content for each. The second chapter provides more detail on context-space mapping, with some additional examples of cross-maps that can be useful in specific types of sensemaking. And finally there's an appendix that lists the various resources referenced within the book.

That's the overall structure of what follows. But the clock's already ticking on this architecture-project: time to get started.

Day 1: Get started

Whatever we do, however we approach it, and whichever part of the organisation we work in, all of enterprise-architecture comes down to one single, simple idea:

- **Things work better when they work together, with clarity, with elegance, on purpose.**

Enterprise-architects are responsible to the organisation to make that happen: the underlying aim of every item of architecture-work in the enterprise is to make things work better for everyone.

And every item of architecture-work should start from an explicit business-question. In this specific case, in exploring the role of architecture itself, the 'business-question' will come from us, but the principle remains the same as for any other architecture task. So for here, the question is this:

- **What do enterprise-architects *do*?**
  And how exactly do they add value to the business?

One obvious driver for value to the business will be speed of response: the work is not going to be of much value if we allow ourselves to get stuck in 'analysis-paralysis'. But the business will also need us to deliver something that is of

practical use: we do need to get the balance right here. So, following an Agile-style development principle, we'll pick an arbitrary but appropriate timescale – two weeks – for a first-level architecture iteration. At the end of that time, we'll review and decide what to do next.

Action: *start a project-diary*. Document the key requirements and decisions to date:

> ***Diary:***
> *Commitment*
> *- use architecture methods etc to describe how to do architecture-development in real-time*
> *- topic for the architecture-project is architecture itself*
> *- document in book-form*
> *- timescale: 10 working days*

This means we're already in Day 1 for the project: no time to waste.

Overall aim, scope and purpose

As a starting-point, we briefly summarise some key themes and understandings about what this work will involve, and ideas about what we want to have achieved by the end of this cycle:

> ***Diary:***
> *Starting-point:*

> *project-stakeholders are architects and architects' clients - use the existing Agile-architecture development-process*
> *- demonstrate the recursion etc within that process*
> *- particularly want to describe the sensemaking and decision-making components of architecture, such as via context-space mapping*

Action: *identify the stakeholders and scope.* Every item of architecture work will apply to and affect one or more groups of stakeholders, so we need to identify who those are, as early as possible in the project. We describe these people as 'stakeholders' rather than 'clients', because although the work is usually *for* a specific group of people – such as you, in this case – there are often many others who will be affected *by* it, and whose feelings and opinions will definitely impact the overall effectiveness of the end-result. It's essential to set the right scope, so it's important to note that for architecture work the scope of *influence* – the stakeholders whose views we need to take into account – is usually several steps broader than the scope of *action* – the part of the business for which we have the authority and budget to enact change.

Action: *identify the methods and overall approach to be used.* The aim here is to use architecture to describe how architecture works, so we'll need to base the work on existing disciplines, frameworks and methods. One key to this is the way in which the same overall practices recur not

just as sequential cycles, but within other cycles – a pattern known as *recursion*. We also want to explore and explain the process of sensemaking and decision-making that is the real core of architecture-practice.

> ***Diary:***
> *Agile-architecture cycle:*
> *1. Setup - context and business-purpose*
> *2. Architecture-side: what do we have, what do we want, what's the difference from here to there*
> *3. Implementation-side: what needs to change, what's the plan, do it*
> *4. Wrap-up: what value was gained, what have we learnt, what's next*
> *Include glossary/thesaurus, models, opportunity/risk, issues etc*

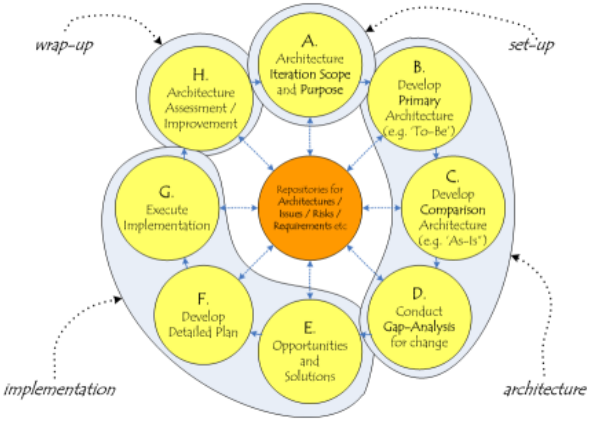


Figure 1: The architecture cycle

As in the project-diary, the architecture-development process is structured as a cycle with four main groups of activities, typically comprising eight distinct phases: a setup phase; a group of three phases on architecture-assessment; another group of three phases on implementing the results of that assessment; and a shared completion-phase that wraps up the overall project (Figure 1).

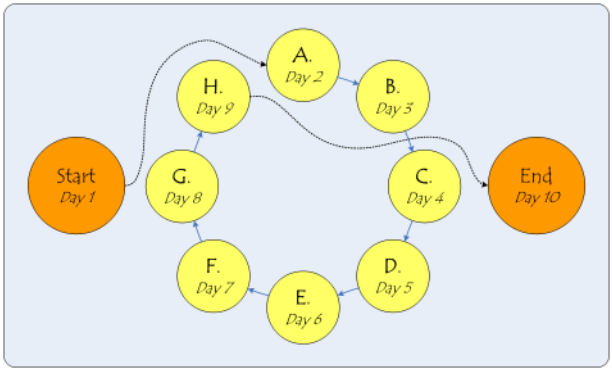


Figure 2: Structure for this project: cycles within a cycle

For this project there will be just one main cycle, although this will often include other cycles within it (Figure 2).

In effect this first day is one very rapid skim through that cycle, looking at purpose, then the needs that arise from that purpose, and what we need to do to action those requirements, followed by a quick wrap-up and review. After that we'll do a full cycle, allocating one day to each set of activities; and then complete the project with a final overall review, which again will in effect be another rapid one-day cycle.

Initial aim, scope and stakeholders

For this initial one-day cycle, the stakeholders are ourselves, the scope is the same "how would we describe

architecture?", and the aim is to develop a plan of action for the remaining work that will deliver useful results in the small amount of time that we have.

Initial assessment

Right at the start of an assessment-phase, by definition, we don't know what we're doing, and we don't know what to do. For many people this can be accompanied by a strong feeling of inadequacy, incompetence, even of failure, so it's important to realise that *this is normal and to be expected at this stage of the process.*

For almost everyone, this kind of inherent uncertainty can be very uncomfortable. And although it takes a lot of practice to become 'comfortable' with being uncomfortable, that's a very useful skill for architects to develop, because our clients will be going through exactly the same experience, and we'll need to help them through it too. What helps most here is to acknowledge what we feel, yet remember to follow the process: keep the focus on the overall aim or 'vision', and then *do something* – almost anything, in fact – to give appropriate ideas somewhere to begin to coalesce.

***Action**: don't fight against the uncertainty, work* with *it.* In this phase of the work it's best to place ourselves in an 'information-rich' environment of some kind, to provide the broadest possible range of triggers for ideas. For some people this will literally be 'noisy' – music, crowds, the

market – whilst others would prefer the library or a wild scatter of papers and images. The key is to keep a notepad or voice-recorder to hand at all times here, to catch the often-fleeting impressions that will start the ball rolling.

> ***Diary:***
> *Assorted notes:*
> *- start with a mind-map of key themes/concerns: what is 'enterprise architecture'? for this purpose?*
> *- how would I use this? - give a real example*
> *- 'what is an enterprise?' - what is 'the enterprise' for this?*
> *- use simple checklists: context-space; five-principles; five-elements; four-dimensions extended-Zachman*

The free-form nature of mind-mapping can be useful here – if only to express how we feel about the uncertainty at this point…

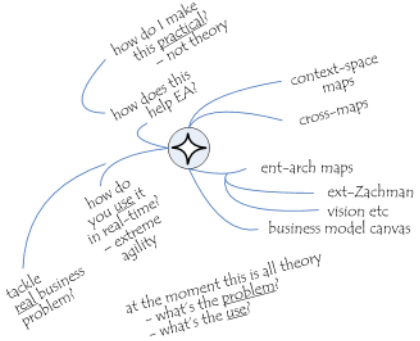


Figure 3: Initial mind-map

Importantly, *all* we are doing at this stage is assessment and information-gathering: we document the ideas and images that come up, but we *don't* take action to follow up on any of them as yet. This does, however, point to a key operating principle that we will use throughout the entire process:

Action: *if anything comes up during a project-phase that more properly fits the function of a later phase, the only action should be to document and tag it for retrieval during that later phase.* For example, that list above includes "how would I use this?" and "use simple checklists" – both of which are more about action than assessment, so we do nothing more about them for now, other than ensure that we will remember them when we get to the 'implementation' stage of this small cycle. But the question "what is an enterprise?" *is* useful for assessment, so we do need to explore that point briefly before moving on.

Perhaps unsurprisingly, the question 'what is an enterprise?' is fundamental to enterprise-architecture. This is important because many discussions about enterprise-architecture will assume that it's solely about IT. The point here is that even if we're only concerned with IT, we still need to set its respective 'enterprise' in a broader scope – *much* broader, in fact.

The key distinction here is that the architecture we develop is *for* an organisation, but *about* an enterprise:

- the *organisation* is bounded by *rules and responsibilities*
- the *enterprise* is bounded by *values and shared commitment*

This essential difference between rule-based versus values-based means that whilst we can sort-of control what happens within an organisation, we can't do the same with an enterprise: the best we can do is negotiate agreements – which is a very different process than issuing organisational edicts…

An organisation is also an enterprise, of course (though an enterprise is not necessarily an organisation) – hence the common habit of describing a business-organisation as 'the enterprise'. But for architecture a useful guideline is that *the enterprise in scope is at least three steps larger than the organisation in scope.*

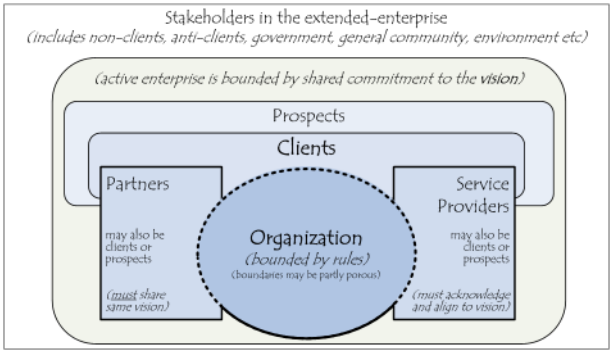


Figure 4: Organisation and enterprise

For a business-organisation, those three steps or layers outward would typically include:

- *layer #1*: partners, suppliers and service-providers
- *layer #2*: clients, prospects, competitors and 'the market'
- *layer #3*: non-clients, broader community and overall business-ecosystem

For a government department or not-for-profit organisation, we might use alternate labels for 'clients', 'prospects' or 'competitors', but the overall structure would be much the same.

And since organisations are often hierarchical, we'll also see the same structure recurring at different levels *within*

organisations. For example, in classic IT-oriented 'enterprise'-architectures such as TOGAF, we would see the following layers:

- *layer #0*: physical IT – TOGAF 'IT-Infrastructure Architecture'
- *layer #1*: the 'users' of the physical IT-infrastructure, namely applications and data and their service or partner interfaces – TOGAF 'Information-Systems Architectures'
- *layer #2*: the clients of those applications, human or otherwise – a rather muddled part of TOGAF 'Business Architecture'
- *layer #3*: the 'business ecosystem' for the overall architecture – the remainder of TOGAF 'Business Architecture'

It's extremely useful to keep that pattern in mind at all times when doing any kind of enterprise-architecture.

Another useful tactic in this phase is to look outside of our own industry. Here, for example, we could turn to Matthew Frederick's checklist-style book on building-architecture, *101 Things I Learned In Architecture School*, and scribble some quick notes in the project-diary on various themes that caught the eye:

> ***Diary:***
> *From the '101 Things' architecture-book:*

- the parti is the central idea for the structure
- what is the parti here?
- sense of place (and what goes on in that place)
- architecture vs space-planning, engineering, design
- "experience of architectural space is influenced by how we arrive"
- value of detail-level disciplines: drawing, lettering, creating shapes
- "good designers are fast on their feet" - an Agile view of the parti
- emphasis on process, not product
- importance of 'thinking about thinking', as meta-methodology
- levels of knowing: simplicity, complexity, informed-simplicity
- static, dynamic, symmetric and asymmetric balance
- design in section, not solely in plan! - views into 'context-space'
- design with models - modelling as a sense-making/design process
- gaining control of design process at first feels like losing control
- architecture an exercise in truth and in narrative - what is the story?
- everything is in context to a larger context - layering of 'enterprise'
- design is constrained by rules, regulations, other people's priorities: use those constraints

to encourage creativity
- -do- something to get started - and give it a name

Remember that there's no particular plan at this stage: it's just about creating a space in which ideas can arise, and collating the results in re-usable form. It may be that in the end we don't use some of these ideas at all: but at this stage we not only don't know but *can't* know which ideas we will or won't use. Yet placing these notes in the project-diary means that they'll be available to us if and when we need them: that's all we're doing here.

One idea from that list that's useful right now is the architectural notion of the '*parti*'. As Frederick puts it, "a *parti* is the central idea or concept of a building … *parti* derives from understandings that are nonarchitectural and must be cultivated before architectural form can be born". It's unlikely to arrive just yet, but we need to keep our awareness open for any pointers to our own enterprise-architecture equivalents of the *parti* – a single unifying theme that will link all aspects of this architecture together.

It's not much, of course, but that's probably all the assessment that we need to do at this very early stage.

Initial implementation

'Implementation' in this context will likewise be very simple: most of it is just an exploration and confirmation of

the key tools that we would use in the main body of the work. For the main project here, these were listed in the earlier notes:

- context-space mapping
- five key principles from systems-theory
- five-elements model of concurrent-lifecycle business processes
- four-dimensions extensions to the Zachman framework

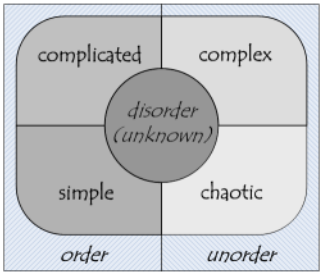


Figure 5: Typical base-frame for context-space mapping

Context-space mapping (CSM) is a method for using descriptive frameworks in architecture. One common example aligns with a well-known framework called Cynefin, that describes a context in terms of four distinct 'domains' of interpretation and action (Figure 5).

It's important at this point to emphasise that context-space mapping is *not* Cynefin, nor is it derived directly from that framework.

The roots of context-space mapping go back to a much earlier book of mine called *Inventing Reality*, first published in 1986. Its core metaphor, derived from work by Snell and others in 1970s, was that reality is a 'swamp' of infinite possibility, within which we have distinct yet often seemingly mutually-exclusive modes for sensemaking and action, such as those of CP Snow's classic archetypes of the scientist and the artist. When I was first introduced to the Cynefin model in the early 2000s, it seemed an almost perfect correlation and confirmation of that previous work: in particular, the four main 'domains' matched almost exactly, and its central 'disorder' region matches the context-space notion of an indeterminate 'the everything'. No surprise, then, that that base-diagram has been a valued part of my enterprise-architecture toolkit ever since.

Unfortunately, though, there are some significant challenges about use of Cynefin for this type of work. Hence throughout this book I've used a different layout – the 'star-diagram' – to emphasise that, unless otherwise stated, what you see here is *not* Cynefin. Context-

> space mapping is *not* the same: they derive from different roots, are used in significantly different ways for different business roles and purposes, and use different approaches to and for validation and verification.
>
> That's probably all that needs to be said here: but it *does* need to be said, and noted as we move on to other context-space maps.

Those four main 'domains' within the star-diagram in effect represent distinct regions in a spectrum of low to high repeatability, or low to high abstraction, with Chaotic at the low end and Simple at the high end. It's simplest to summarise them as follows:

- *Simple*: 'order', decisions in real-time, based on simple true/false logic applied to simple cause-effect relationships with very high repeatability
- *Complicated*: 'order', decisions either before or after the event, based on analysis of complicated but linear cause-effect relationships with high repeatability
- *Complex*: 'unorder', decisions either before or after the event, based on iterative experiments with non-linear cause-effect relationships that have only partial repeatability
- *Chaotic*: 'unorder', decisions in real-time, based on principles and values, in contexts with no discernible cause-effect relationships and low to no repeatability

We also need to be aware of the undefined 'the everything' that exists before any sensemaking takes place, and before any level of repeatability can be identified: this is represented in the star-frame diagram by the central region of *disorder*.

The *five systems-theory principles* provide an essential checklist of patterns to watch for in enterprise-architectures:

- *rotation*: a systematic process to assess a context from multiple yet related perspectives – such as a checklist or overview-diagram
- *reciprocation*: processes that create balance between systems or between components in a system
- *resonance*: positive-feedback or feedforward, which increase the 'snowball effect' towards self-propagation, or negative-feedback or damping, which diminish the effect
- *recursion*: relationships or interactions which repeat or are 'self-similar' at different scales – such as the classic hierarchical org-chart
- *reflexion*: holographic inverse of recursion – the whole is reflected in, and can be identified within, any part at any scale

Reflexion is perhaps the strangest aspect of systems-theory, yet one of the most valuable in enterprise-architecture.

Through it we see that everything is connected to everything else, but is also *part of* everything else. A useful analogy here is a hologram: unlike an ordinary photograph, even the tiniest fragment of a true hologram will always contain a complete picture of the whole.

The five-elements model applies those principles of recursion and reflexion to any type of lifecycle in the enterprise:

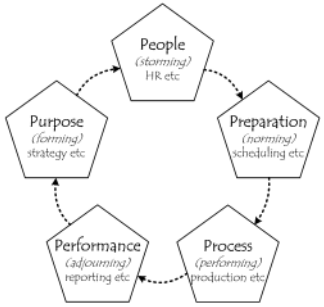


Figure 6: Five-elements lifecycles model

This is actually the same sequence as used in the classic Group Dynamics project-lifecycle – Forming, Storming, Norming, Performing, Adjourning – though here it is more generic, applying not just to projects but to relationships between different areas of the business, such as strategy, HR, scheduling, production and reporting respectively. The phases in the lifecycle also align well with the key dimensions of *effectiveness*:

- *Purpose*: focuses on *appropriateness* in the enterprise

- *People*: addresses *elegance*, values, simplicity, ergonomics and other human-factors in the enterprise
- *Planning*: emphasises *efficiency* – making the best use of the enterprise's available resources
- *Process*: ensures *reliability* and availability of the functions and services required by the enterprise
- *Performance*: assures overall *integration* between all the different elements of the enterprise

And the multi-dimensional *extended-Zachman framework* provides a means to categorise anything we come across in our modelling of the enterprise. The classic Zachman taxonomy is well-known to enterprise-architects, and consists of a simple row/column grid. The extension adds an extra row at the top, and inserts an extra dimension of 'segments' which categorise *types* of entities (Figure 7).

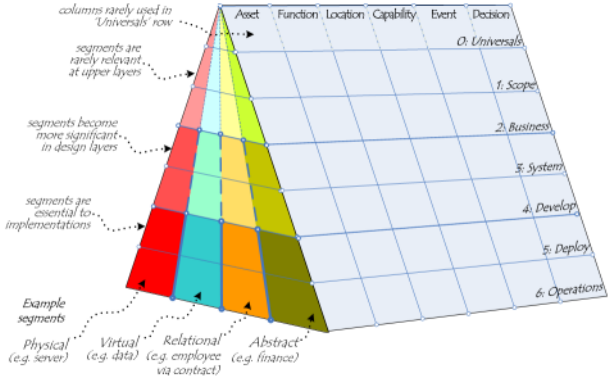


Figure 7: Framework rows, columns and segments

As in the original Zachman framework, the rows here represent types of responsibilities or viewpoints:

- *row-0*: ***'Universals'*** – core constants to which everything should align – the key points of connection with enterprise partners and other stakeholders
- *row-1*: ***'Scope'***– adds possibility of change: key 'items of interest' in each category, *without* relationships
- *row-2*: ***'Business'*** – adds relationships and dependencies between entities: core entities described in business-terms
- *row-3*: ***'System'*** – adds attributes to abstract 'logical' entities, expanded out into *implementation-independent* designs

- *row-4*: ***'Develop'*** – adds details for real-world 'physical' *implementation-dependent* designs
- *row -5*: ***'Deploy'*** – adds details of intended future deployment as actual software, actual business-processes, work-instructions, hardware, networks etc
- *row-6*: ***'Operations'*** – adds details of actual usage: specific instances of entities, processes etc, as created, modified, and acted on in real-time operations

The 'Universals' row actually represents a separate dimension, a kind of backplane to which *everything* must connect; it's simpler, though, to show it as if it's an extra row on the frame.

The columns of the grid are slightly different from the Zachman standard, to correct some design-inconsistencies in the original:

- ***assets*** ('What'): physical objects, data, links to people, brands, finances, etc
- ***functions*** ('How'): activities or services to create change, distinct from the agent (machine, software, person etc) that carries out that activity
- ***locations*** ('Where'): physical (geography etc), virtual (`IP` nodes, `http` addresses etc), relational (social networks etc), time

- ***capabilities***, often as ***roles*** or 'actors' ('Who'): human, machine, software application, etc, and either individual or collective
- ***events*** ('When'): physical, virtual, human, business-rule, time-based or other event
- ***decisions*** ('Why'): reasons, constraints and other tests which trigger or validate the respective condition, as in strategy, policy, business-requirements, business-rules, regulations etc.

The 'segments' add an essential dimension that is either implied or missing entirely from the Zachman original:

- ***physical***: tangible assets, mechanical processes and functions, physical or temporal locations, physical events; also Simple-domain ***rule-based*** capabilities and decisions
- ***virtual***: intangible assets such as data, software processes and functions, logical locations, data-driven events; also Complicated-domain ***analytic*** capabilities and decisions
- ***relational***: links to people (as indirect 'asset'), manual processes and functions, social/relational locations, human events; also Complex-domain ***heuristic*** capabilities and decisions

- ***aspirational***: abstract assets such as principles, values, brands and belonging, morale and self-belief, locations within value-webs, some business-rule events; also Chaotic-domain ***principle-based*** capabilities and decisions

There are also a few additional uncategorised segments such as for financial assets and functions, energy as an asset, and time as an event-trigger.

These segments represent what are actually distinct *dimensions* within context-space, and are fundamentally different from each other in scope and function. For example:

- *physical assets* are 'alienable' – if I give it to you, I no longer have it
- *virtual assets* are 'non-alienable' – if I give it to you, I also still have it
- *relational assets* exist *between* two entities – if either party drops the relationship, it ceases to exist
- *aspirational assets* represent relationships that are more a one-sided 'to' rather than a balanced 'between'

Most real-world entities we deal with in enterprise-architecture are composites that straddle across multiple columns and/or segments. For example, a book is an asset that is both physical and virtual – object and information; a service is

a merging of function and capability in which assets may be changed in accordance with decisions and events; and a business-model straddles every row, column and segment of the entire frame.

Once we've identified the tools and techniques that we will use for the main cycle, we then need somewhere to store all of the information that's required:

Action: *define and implement information-stores* or 'repositories' for architectural information, including glossary/thesaurus, stores for models and project-management information, and registers for risks, opportunities and other issues – see chapter *The architecture information-stores*.

That covers most of the 'implementation' needed for this initial cycle. The project-diary for this stage includes some additional notes, most of which will carry over to the next cycle:

> ***Diary:***
> *More random notes and jottings:*
> *- architecture is often about the 'non-functional', the qualitative*
> *- is about development of judgement and awareness (otherwise all we have is 'follow-the-rules', which destroys differentiation)*
> *- use method and repositories to illustrate themselves*
> *- OODA (observe, orient, decide, act) inside-loop in sense-making*

- serendipity, obliquity
- CSM: order = direct approach, unorder = oblique approach
- also CSM: in real-time we don't have time for analysis or experiment - everything that we do is the analysis or experiment
- decisions made in real-time may have impacts that last for decades
- dynamics of sense-making - 'go for a walk' through context-space
- content, context, connections, purpose

But as we noted in the previous stage, we'll also need some kind of concrete project-example, to ensure that we don't get lost in the abstract, solely discussing 'the architecture of architecture'. As shown in the project-diary, the example that came to mind here was a real enterprise-level concern about respect:

***Diary**:*
Real client business-problem as demonstrator
- loss of respect: "we've gone from the most-respected bank in our region to least-respected
- what can we do about this?"

So we'll use this as a parallel worked-example throughout the main project-cycle.

Wrap-up on initial cycle

Every architecture-cycle should end with a brief review. At first we may seem not to have done much in this day's work, and it's quite likely there'll still be a lot of uncertainty about what we're aiming to do and what we'll achieve by the end of the overall project. But in fact we've now done most of the essential groundwork that will underpin all of the subsequent tasks – and as with all foundations, there's not much that will show on the surface! The value of that groundwork will become more evident as we move through the main architecture-cycle over the next few days.

Application

- How are architecture-projects set up in your own business context? What information do you receive with the work-request? What guidelines and metrics – if any – are defined so as to enable benefits-realisation assessment on project completion?
- Do you work on your own, as part of a team of architects, or a more diverse team that includes people from a variety of organisational functions? If you work on your own, what options and actions do you have for peer-review? If you work as part of a team, how do you manage the group-dynamics and the respective roles and responsibilities?

- How do you identify the stakeholders for the project? The immediate clients for the project will usually be obvious, but which other stakeholders' interests need to be addressed, and how?
- What, for you, is the typical timescale and scope for an architecture project?
- What methods do you use to provide guidance and governance for each architecture project? In what ways do these methods change for different timescales and scopes?
- How do you start a project? What assessment takes place before you begin, and immediately after starting? How do you manage the inherent uncertainties at the start of each project?
- Which frameworks, tools and techniques do you use in your work? How do you select these, and why? And do they differ from project to project?
- What information do you collect during architecture work? How do you store, manage and maintain this information?

Day 2: Purpose, scope and context

At this point we start a new architecture-cycle, for which we've allocated eight of our ten available days: one day for each phase of the standard cycle. This will allow us to explore in more depth our 'one idea' that things work better when they work together, on purpose. We summarised the basic structure of the architecture-cycle in the previous day's overview, but we now need to flesh out a bit more of the detail:

- *Phase A*: Define business-scope, business-purpose and time-horizon(s) for the iteration; scope also identifies respective stakeholders and applicable governance for assessment and any probable implementation phases
- *Phase B*: For the primary time-horizon ('as-is' or 'to-be'), identify the baseline of what is already known in the architecture-repositories about the scope; then assess the context in more depth, adding content to the repositories as we do so
- *Phase C*: Repeat Phase B for the one or more comparison time-horizons ('to-be', 'as-is' or intermediates) specified in Phase A.
- *Phase D*: Do a gap-analysis for each 'as-is' and 'to-be' pair (from Phases B and C), to identify requirements,

constraints, risks, opportunities and suchlike for future change.

- *Phase E*: Review the results of Phase D to allocate priorities to requirements and identify appropriate means to implement the requisite changes or 'solutions'.
- *Phase F*: Establish a detailed plan to handle the changes 'from here to there' – in particular, dealing with the 'people' and 'preparation' aspects of change.
- *Phase G*: Architecture assists change-governance with compliance, consistency and inter-project synergies during implementation of the planned business change.
- *Phase H*: Return to architecture-governance to do a 'lessons-learned' review in relation to the respective business context, and identify any needs for further related architecture work.

The task for *this* phase – Phase A - is to identify and document the key themes and decisions for the architecture-cycle. That's what this day will address. Much of this is administrative, setting the scope of the project and dealing with authorisations and paperwork – but we still need to think in architectural terms at all times.

Main project: 'the architecture of architecture'

There was a lot of effort that went into the previous day's setup-work, as the project-diary observes at this point:

> ***Diary:***
> *Running a bit behind schedule - still playing catch-up to yesterday*

But often key ideas will come up when we least expect them – and that's why the project-diary is so important, as a means to catch those ideas as they pass by. In this case it was a crucial cross-reference from William Beveridge's scientific classic *The Art of Scientific Investigation*:

> ***Diary:***
> *from Beveridge intro:*
> *"Elaborate apparatus plays an important part in the science of today, but I sometimes wonder if we are not inclined to forget that the most important instrument in research must always be the mind of [the researcher].*
> *"It is true that much time and effort is devoted to training and equipping the scientist's mind, but little attention is paid to the technicalities of making the best use of it.*
> *"There is [at present] no book which systematises the knowledge available on the practice*

> *and mental skills - the art - of scientific investigation."*

In essence this is the exact same concern that we're dealing with here in enterprise-architecture: "It is true that much time and effort is devoted to training the [architect's] mind", in terms of the available frameworks and methodologies and so on, "but little attention is paid to the technicalities of making the best use of it – the *art* of [architectural] investigation". That focus on 'the technicalities of making the best use of the architect's mind' should probably become the central theme – the *parti* – of our main project here.

So let's get this iteration of the cycle started, doing it step by step. The full detail for the method is described in the companion-book *Bridging the Silos*, which tells us that the objectives of Phase A are:

- establish the key business concerns and constraints;
- identify iteration scope, components and priorities;
- identify the stakeholders, and their concerns and objectives;
- identify business principles, goals and strategic drivers;
- understand mutual impacts of other enterprise architecture development-cycles going on in parallel;
- ensure that we have the authority to do the work;

- secure resources and formal approval to proceed.

Step 1: Identify purpose and scope of architecture cycle

This purpose should always be described in *business* terms, and should *not* presuppose any particular solution, as another note in the project-diary confirms:

> ***Diary:***
> *from peer-review meeting with Kevin S:*
> *- don't start from 'solutions'! - spend the time working on the problem-domain, the solutions needed will arise naturally from that*

If we come across any ideas for solutions, they go into the project-diary for review later – but not here, and not now.

Here we need to define what is and is not in scope for this specific effort. The book says that we should establish the breadth of coverage, the level of detail, the architecture domains, respective time-horizons for 'as-is' and 'to-be' of the enterprise for the cycle, and any existing assets that we're likely to re-use. And for a formal project we should record the results of this step in a first draft of a document called the 'Statement of Architecture Work'. For this, though, we can simplify everything right down:

- *business-purpose*: use architecture to explain and enhance the way we do architecture
- *sponsor (primary stakeholder)*: us

- *breadth of coverage*: mainly architecture itself, but also anywhere that architecture affects
- *level of detail*: anything we can usefully cover in the time available
- *architecture domains*: it's mostly about architecture itself
- *time-horizons*: as-is = now, to-be = ten days from now; to-be first (to-be as primary, as-is as comparison)
- *asset re-use*: whatever architecture tools, techniques and methods that we already have to hand

We don't need a formal Statement of Architecture Work for this – we can document it instead in the project-diary.

Step 2: Identify and review applicable principles, policies etc

These should be straightforward:

- *policies*: whatever we already have that applies to architecture in general in this work-environment

If they don't exist, that's something that we'll need to work on as we go through this cycle. We place a note in the project-diary to document this, anyway.

Step 3: Identify business goals and strategic drivers

This is about where this item of work will fit within the broader picture of the enterprise. In a normal architecture-cycle we may need to do some chasing-around at this point to find out what they really are – because often the immediate client won't know, and may not even care. But for our purposes here, these again will be simple and straightforward:

- *goals*: enhance skills and capabilities of the architecture team
- *drivers*: enhance overall effectiveness of the enterprise

These go into the project-diary too.

Step 4: Establish architecture-framework scope of cycle

This uses the extended-Zachman framework that we explored briefly in the previous day's work. In reality it's just a large checklist that we can use to do a first pass through the 'problem-space', to give some initial suggestions about what is and is not in scope from an architectural perspective. Almost all real-world entities are made up of 'composites' that straddle the framework's rather simplistic categories – the 'primitives' or framework-cells – and we do need to remember at all times that those entities *are* actually composites.

To guide our sensemaking and redesign, we need to end up with abstractions such as the 'architectural primitives' of

the framework. But we always *start* from the composites in the real world – all the things that we see, that we touch, with which we interact – and then run the usual design process backwards, from complete design back to their underlying components. Doing this will also help us to identify the models that we'll need in the later phases when we get deeper into architectural assessment.

In this case, we're not looking at the outcomes of architecture, but at the skills and capabilities that are needed for architecture itself. From the framework perspective, this makes the main focus of the scope very simple:

- *primary scope*: layers R3 ('system') to R5 ('deploy); capabilities to tackle all types of problems; implemented by people (via 'relational assets')

We then work across the columns to assess what else might come into that scope:

- *assets*: quite a bit of information (virtual assets) and links to people (relational assets)
- *functions*: it's more about capabilities, but we might need to consider how those capabilities link together with functions as architectural services
- *locations*: anywhere that architecture work takes place – physical, virtual and/or relational

- *capabilities*: particularly the other capabilities needed for some of our support-services
- *events*: mainly relational 'people-events', though others may come into scope as we move closer to real-time architecture
- *decisions*: many different types at every different layer – that's really what we're working on in architecture

As usual, all of these go into the project-diary for later reference, particularly during the architecture-assessment phases that are coming up next.

Step 5: Identify other stakeholders, concerns, requirements

The client or sponsor is the primary stakeholder – and in this case that's us. But we also need to identify anyone else who may be affected by the results of this item of architecture work, both within the organisation and the broader enterprise. In essence that's everyone, of course, but we can usefully split this into first- and second-order additional-stakeholders, in much the same way that we might partition a supply-chain from supplier's supplier through to customer's customer:

- *first-order*: strategists; change-managers; project-leads; project- and program-managers; other architects and system-integrators

- *second-order*: operations-staff (especially those doing front-line innovation); developers; other managers; other key players beyond the organisation

It *is* important to remember that these might be anyone at all: we won't often work with the organisation's end-customers, perhaps, but they are definitely 'stakeholders' of the results of our work.

Later on, if we organise the architecture information-repository around the structure of the framework, we can link people and their responsibilities to each of the items that we've recorded in the repository. By defining scope in terms of the framework, as in the previous step, we also identify many of the probable stakeholders for architecture-work. We should also review the issues-register, risks-register and our other architecture information-sources for other potential stakeholders whose concerns may be impacted by the project. It probably isn't all that important for this project, but for other more far-reaching projects it can save a lot of heartache – and prevent a lot of angry calls from stakeholders who didn't appreciate being left out of the discussion…

We record all of this too in the project-diary.

Step 6: Identify additional requirements

For a larger project there might be some additional organisational or enterprise-wide limits on time, schedule, resources or the like. For this, probably the only real constraint is time:

- *available time*: ten working days

We add that item to the list in the project-diary.

Step 7: Finalise plan and secure approval to proceed

For a formal project, we would need here to add a lot more to the Statement of Architecture Work: define a plan of architecture activities that will address all the requirements, within the scope and constraints, conforming with the business and architecture principles, and so on. We would also need to estimate the resources needed, and perhaps develop a roadmap and schedule for the proposed development. We would need to document all of those items in the Statement of Architecture Work for the project, and present it for formal review, before asking for authority to proceed.

In this case, though, the plan is already set – we're simply going to walk through the architecture process to review architecture itself – and we don't need anyone else's authorisation to do that:

- *plan*: use architecture to review architecture
- *authorisation*: none needed

Once we've documented that in the project-diary, we would be ready to proceed to the first part of the assessment.

Example project: 'respect', for a bank

To illustrate each of themes in the main project, we'll run another real-world project in parallel.

> This example-project is actually a composite drawn from several real business-transformation assignments during the past year. For obvious reasons of confidentiality, many of the key details here have been changed, or combined from different organisations, but the issues and background described here are essentially equivalent to those in the originals.

We're contacted by the organisational-development manager of the largest banking group in this country, the regional arm of a global corporation. They have a problem, he says, and want to know how enterprise-architecture would help. The project-diary summarises that first meeting:

> ***Project-diary:***
> *client-meeting (change-manager):*
> *- key issue is respect: "we've gone from the most-respected bank in our region to the least-respected - what can we do about it?"*
> *- consequences: loss of trust from government, active rejection in market, loss of market share*
> *- short-term profits are okay - which keeps*

parent-group at bay for now - but can see profits collapsing in near future if nothing is done
- whole-of-organisation scope but limited authority for change - will need CEO's full backing to make it work
- take-over of another bank last year - still working on integration
- feeling the effects of the worldwide credit-crunch

It's urgent, he says, yet the organisation is already struggling from 'change-fatigue' arising from that take-over, and funds are very tight at present both from that and from the overall malaise of the banking sector worldwide. Their own 'enterprise-architecture' unit only covers IT concerns, and is still at a fairly early maturity-level, so will not be able to do the work. He needs something that he can weave into his existing change-programmes at minimal cost and with minimal disruption; he wants concrete suggestions on that from us within two weeks at most.

Two key points came up in that meeting: they're able to think beyond just the short-term – which many organisations don't – and they can see beyond surface symptoms to deeper causes. Both of those bode well for a true enterprise-scope architecture, which is clearly what this will need to be. What's not so good is that our client is the change-manager, but whatever we specify will need the CEO's full support – yet we're told he's a 'numbers-man' whose

main focus is the quarterly figures, so we may have a real problem right there.

This isn't a large-scale project, though, so we can run it with only minimal governance, using a project-diary rather than a formal Statement of Architecture Work. But we still need all of the details to define what this project will be, so we go through the standard project-start checklist:

Step 1: Identify purpose and scope of architecture cycle

We can derive most of this direct from the information we have so far, though one important question comes up in the project-diary:

> ***Project-diary:***
> *which way round for assessment: as-is first, or to-be? - the to-be is actually more about recreating the conditions of the past*

The to-be is relatively easy to describe, so we decide to do the as-is assessment first, because that's where the known problems are. The outcome of this project will almost certainly call for cultural changes, for which the usual guideline is that these take several years to embed: so we'll probably need to specify for intermediate time-horizons as well as for the final desired 'future state'.

Given that, we can now summarise the overall project:

- *business-purpose*: restore community/market respect lost by the organisation

- *sponsor (primary stakeholder)*: change-manager
- *breadth of coverage*: high-level overview, organisation-wide, extending outward into extended-enterprise (including community and government)
- *level of detail*: anything we can usefully cover in the time available, with an emphasis on the 'respect' theme
- *architecture domains*: emphasis on business-architecture, but may extend downward into IT or other detail-level domains
- *time-horizons*: to-be = three years from now, with probable intermediates at six months and one year
- *asset re-use*: internal documents, publicity material, external surveys, customer-satisfaction and staff-satisfaction surveys

We document all of this in the project-diary.

Step 2: Identify and review applicable principles, policies etc

The banking industry is subject to many rules and regulations, with international, national, local or organisational scope; but most of those apply at the detail-level rather than at the whole–enterprise levels that we'll need to work with here. We're also working independently from the internal architecture-unit, so for *this* part of the work the only governance-policies we'll need are our own:

- *policies*: general architecture-governance

We'll need to identify applicable *implementation*-governance as we do the assessment, though. We note all of this in the project-diary.

Step 3: Identify business goals and strategic drivers

As outlined in that client-meeting, the initial goals and drivers are as follows:

- *goals*: enhance respect of the bank – both from others and within itself – to at least the levels enjoyed a few years ago
- *drivers*: market credibility; government and community relations; medium- to long-term profitability

These go into the project-diary, together with a note that other goals and drivers may arise during the assessment.

Step 4: Establish architecture-framework scope of cycle

This may be quite difficult to describe, because clearly 'respect' is an issue that pervades the entire organisation and enterprise. When respect fails, it's usually because the organisation has lost track of its business-purpose, which would take us right up to the row-0 'Universals'; but it's also about actual business-practices, all the way down to the framework's row-5 or row-6. We make this manageable, though, by remembering that we only need enough to build a quick 'holograph' overview of the issues,

not Zachman's 'excruciating detail' about everything… The real emphasis needs to be on anything that affects the organisation's links with people, both externally and internally:

- *primary scope*: layers R0 ('universals') to R5 ('deploy'); relational assets (as links with real people)

Working across the columns will suggest other themes in scope:

- *assets*: mostly relational assets, also virtual assets (information)
- *functions*: any functions that change or impact on relational-assets
- *locations*: any location – physical, virtual and/or relational – that impact on relational assets
- *capabilities*: any capabilities that impact on relational-assets
- *events*: mainly relational 'people-events'
- *decisions*: any decisions, business-rules etc that impact on relational-assets

All of these go into the project-diary for reference during the architecture-assessment phases.

Step 5: Identify other stakeholders, concerns, requirements

The direct client here is the change-manager, who is an important stakeholder who'll be in charge of implementing any proposals that come out of this process. But within the organisation, *the* key stakeholder is the CEO, because that's who holds the ultimate responsibility for the organisation's 'universals', and hence whose concerns and needs we most need to satisfy. Behind the CEO, of course, are the CEO's equivalents in the parent-corporation; and behind them the anonymous shareholders, whose benefit is nominally the highest priority for the corporation. Those who would be engaged in implementation of change would be viewed somewhat separately, giving us a stakeholder 'stack' as follows:

- *priority*: CEO, executive, parent executive, shareholders
- *first-order*: strategists; change-managers; project-leads; project- and program-managers; other architects and system-integrators
- *second-order*: operations-staff (especially customer-facing staff); developers; other managers; other key players beyond the organisation

Yet beyond all of those, as indicated in that earlier diagram of organisation versus enterprise, are all the *indirect* stakeholders that the organisation must engage with in order to create the desired 'shareholder-value'. Their respect – or

lack of it – is our actual focus here, so we *must* include them in our list of stakeholders:

- *indirect-stakeholders*: clients, prospects, non-clients, anti-clients, government, general community

This list of stakeholders goes into the respective section of the project-diary.

Step 6: Identify additional requirements

For this brief project, the key constraint is time – both our own, and those of the people with whom we'll need to engage. We will only be making recommendations, not doing implementations, so it'll be straightforward enough to identify funding- and resource-requirements up-front:

- *schedule*: ten working days (elapsed time)
- *client staff-availability*: key stakeholders (for information-gathering)

We add these items to the list in the project-diary.

Step 7: Finalise plan and secure approval to proceed

The client needs a written proposal that he can take to the CEO for approval: once that's signed off, we're ready to go:

- *plan*: review available materials; run two workshops (one for executive, one for representatives of customer-facing staff); derive strategy; deliver recommendations and proposals

- *authorisation*: client (change-manager) for funding, CEO for authority to proceed

This completes the start-up summary documented in the project-diary.

Application

- How do you start up a new architecture-project? From where do you obtain information about the business-problem, the scope, stakeholders, applicable policies and the like? How do you ensure that you have the requisite authority to do the work – especially if you have to cross silo-boundaries to do it? Who funds and authorises the work?
- What frameworks and processes do you use to guide planning for architecture-projects? How do you estimate schedules, costs, resources-needs and the like?
- When architecture-preparation highlights potential political issues – particularly concerns around authority or 'turf' – what planning do you need to do to mitigate those risks?

Day 3: What's going on?

At this point we begin the architectural assessment proper, to find out more about how 'things work better when they work together' within architecture itself. We start with the 'primary context' – the desired or actual context at the time-horizon we chose in the previous phase as the point to or from which we would construct our roadmap of 'from here to there'. Usually we would do the 'to-be' assessment first, but in some cases – and our example-project is one of them – it's better to start with the 'as-is'.

Our other objectives for this phase are:

- select relevant architecture viewpoints that will enable us to demonstrate how the stakeholder concerns are addressed in the overall architecture;
- select the relevant tools and techniques to be used in association with the selected viewpoints.

If we were to do this 'by the book', the process-steps would be:

- develop baseline-architecture for primary context
- select reference-models, views and viewpoints
- create and update primary-context architecture models

- review primary-context architecture against qualitative criteria
- finalise building-blocks for the architectural scope
- conduct checkpoint-review for stakeholders

The reality, of course, is rarely as simple as that. In principle those *are* the steps we need to follow; but in practice it's almost never the neat straight-line sequence shown in those idealised process-diagrams, but something more like a ball of wool after the kitten has chased it across the floor a few times. We use the term 'iterative' more as a euphemism than anything else: 'chaotic mess' might be a more accurate term, given how it often feels…

In the early stages especially, the one thing we'll discover is that much if not most of what we've been told – or will be told – will turn out to be out of date, or incomplete, or just plain wrong. Seeming certainties often aren't. Every stakeholder has their own views, which each usually turn out to be only one side of a much more complex story: as Edward de Bono once put it, "everyone is always right, but no-one is ever right". And somehow we have to make sense from all of this, and derive something out of it that others can *use*. But that *is* our task here: it's *our* responsibility.

The one most important danger is something we'd already noted in the project-diary:

Diary:

Don't start from 'solutions'! – focus instead on the problem-domain

Much of the time, though, that injunction against solutions is not quite right. We do *need* a real usable solution at some point: the danger is about *premature fixation* on any putative solution, rather than all solutions as such. In practice we'll often need to create a temporary 'solution' to give our stakeholders something to argue about and tell us that it's wrong – which it probably is, at first. But that 'wrongness' then gives something else to test, iterating towards a solution that *does* do what our stakeholders need.

Yet the point here is that during most of this iterative process we'll be 'in the wrong' – and many of our stakeholders will be very quick to tell us so, too. Which is not pleasant – but that's what the work demands. In fact there's a very simple test here: *if it doesn't feel uncomfortable, we're probably not doing the job properly.* That's something to think about whilst we're working, anyway.

Main project: 'to-be' assessment of architecture

So: on to assessment for the main project, which we summarised in the project-diary as follows:

Diary:
business-purpose: use architecture to enhance

> *architecture practice*
> *sponsor (primary stakeholder): us*
> *breadth of coverage: architecture, plus anywhere architecture affects*
> *level of detail: anything we can usefully cover in the time available*
> *architecture domains: mainly architecture itself*
> *time-horizons: to-be as primary, as-is as comparison*
> *asset re-use: available architecture tools, techniques and methods*

For this we'll be doing the 'to-be' in this phase, and move back to the 'as-is' – our current skillsets – in the subsequent phase. But how *do* we do this assessment? The project-diary records the sense of frustration and uncertainty here:

> ***Diary**: how -do- we use architecture to assess the architecture of architecture? feels like spinning in circles – no traction, no place to start…*

The short answer is 'follow the process': do it by the book, *but be ready to do something else at any time*, as long as it seems to make sense within the sense of the whole, then return to the structure of the process once that moment of certainty is lost.

Step 1: Develop baseline architecture for 'to-be' context

Some architecture methods use the term 'baseline' in a somewhat different way, but for our purposes here the baseline is the description that we have for this architectural scope *prior to any assessment.* We create this baseline from whatever information that we *already* have about this context in our information-stores – models, requirements, the risks, opportunities and issues registers, glossary and thesaurus and so on. We defined the applicable items earlier in the project-diary:

> ***Diary:***
> *primary scope: mainly layers R3 ('system') to R5 ('deploy); capabilities to tackle all types of problems; implemented by people*
> *assets: information and links with people*
> *functions: how architecture skills and capabilities link into services*
> *locations: may be physical, virtual and/or relational*
> *capabilities: architecture-capabilities and support-services*
> *events: relational 'people-events', others for real-time needs*
> *decisions: many different types at every different layer*

It's quite probable that we don't have anything about architecture itself in the information-stores, because most people won't think of it as a subject for architecture. But

if we do have any, we note that information, and perhaps build a small set of views and reference-models if it seems useful. In any case, we'll only need to do this once, as a known baseline to return to if we get lost during the various iterations of the assessment.

The baseline should always include any overarching enterprise-wide principles, standards and the like from the framework's 'Universals' row, as identified during the previous phase. One example would be to summarise what architecture *is* and *does*:

- it's a *body of knowledge* about *structure and purpose*
- it's used in *decision-making* throughout the enterprise
- it's used to guide *designs*, of any type, that would contribute in *practical and effective ways* towards the *aims* of the *organisation* and *enterprise*

We'll use that as an initial baseline.

Step 2: Select reference-models, views and viewpoints

Although there are plenty of frameworks and models already available for use *in* architecture, currently there are no standard reference-models for enterprise-architecture itself. The nearest that exist are the various certification schemes, which are present are suitable more for IT- or software-architectures than for a complete enterprise-wide scope. (The TOGAF 9 specification does include a summary

of skillsets for architecture, but again most of those are for IT-architectures only.) So if there are no standards, we'll make do with those we listed earlier:

- five-domain context-space mapping
- five systems-theory principles
- five-elements lifecycle
- extended-Zachman framework

That should give us enough to start with for now.

Step 3: Create and update 'to-be' architecture models

Here we expand the baseline architecture into a comprehensive architecture for the iteration context – in other words, the architecture of architecture.

We won't do this in much detail here – just enough to get started, because the real changes will occur as we apply it in practice over the coming days, months and years. What we look for are ideas or themes that would describe our 'architecture of architecture', and core *requirements* for that capability within the enterprise.

This aspect of assessment is mainly about *sensemaking*, and is also going to be iterative and somewhat chaotic – so at the start we deliberately *allow* it to be 'chaotic', a free flow of ideas, and then allow some kind of structure to emerge from there.

Let's start with *context-space mapping* using the star-diagram:

- initially there is only Disorder, 'the unknown'
- we begin sensemaking in Chaos to collect new information, iterating back-and-forth between there and Complex to identify usable patterns
- we iterate between Complex and Complicated to derive designs
- we simplify designs – especially for software, or anything for real-time –by iterating between Complicated and Simple
- **but** we never forget that the real world is actually Disorder

We could summarise this visually as follows:

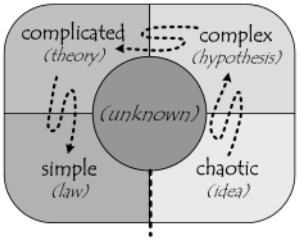


Figure 8: Context-space mapping: iteration between domains

This also clarifies the crucial distinction between architecture and design. They're actually flip-sides of each other, but architecture faces towards the big-picture, the abstract, the overall aim or purpose, whilst design faces towards the detail, the concrete, the practical. On its own, architecture does almost nothing – it's only when it is literally 'realised' through design that it becomes useful. Yet it's also where we're forced to be honest that everything we do is actually a subjective *choice* – whereas design can often pretend to be 'objective', because by the time we reach the design stage we're already following predefined rules. We could summarise this in another diagram:

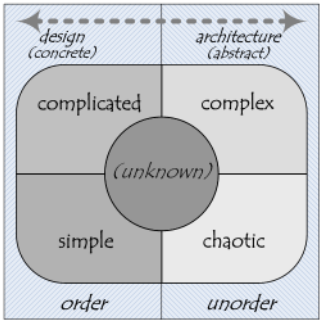


Figure 9: Architecture and design: unorder versus order

The outcome of this is that we need our practices to reflect where architecture sits within the overall functions of innovation and change: it's clear that it must have a large element of *sensemaking* as a precursor to design.

Next, compare against the *systems-theory principles*.

First of these is *rotation*, which is clearly central to much of our work: we rotate between multiple views and viewpoints, or work our way through checklists. We will want to use this both in any architectures we create, and in the processes of architecture itself. In terms of context-space mapping, it's a Simple-domain technique: quick and easy, but with the risk that we have no certain means within the technique itself to assess whether the checklist is complete, or is the right one to use in that context. So whilst we will collect many different 'rotations' for our architecture toolkit – another addition to our requirements-list – we would also need other techniques to select the appropriate ones for each context.

The architecture-development process is another 'rotation', in that it provides a step-by-step sequence to follow – though note that it in effect defines a *default* set of steps, rather than the sequence that we would actually follow in practice, so again other techniques would be needed to decide when to deviate from the default path, and when to return to it.

The next systems-theory principles are *reciprocation* and *resonance*, which in practice act as a matched pair, mainly in the Complicated domain. These will form a much-valued part of the architectural assessment toolkit, particularly to model dependencies, feedback-loops and delays to optimise balance and effectiveness across the enterprise. But they may not apply that much *within* architecture itself: it might be useful to model some of the loops and delays in the architecture process, but that's probably all that we would

do.

The final pair of systems-theory principles, *recursion* and *reflexion*, are fundamentally important to architecture itself – perhaps a key part of what distinguishes architecture from design. In terms of context-space mapping they sit mainly in the Complex domain, though also spread to other domains from there – for example, one of the key benefits of using recursion is that it can make designs a great deal simpler. Architecture itself is highly recursive, applying the same basic principles in every part and at every level of the context and enterprise, whilst the architectural notion of the unifying *parti* represents a highly-desirable example of reflexion.

Next, compare against the *five-elements lifecycle map*. Architecture itself is primarily focussed on the dynamic relationships between structure and purpose, which tells us straight away that there'll be an emphasis on the Purpose and Preparation phases of the lifecycle. This in turn indicates the need for a great deal of attention on the People phase that acts as the bridge between them – which is what we see in architecture practice, of course.

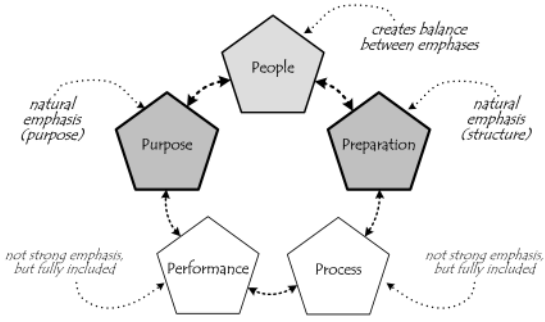


Figure 10: Architecture emphases in lifecycle

Architecture has its own Process activities, though is not much involved in those of others elsewhere in the enterprise; but there would need to be significant attention paid to 'bottom-up' themes coming *from* the production contexts, and also to metrics and the like both for and from the Performance phase. Note too that this five-element lifecycle is also an important example of architectural recursion – the same elements repeat at every level and in every context.

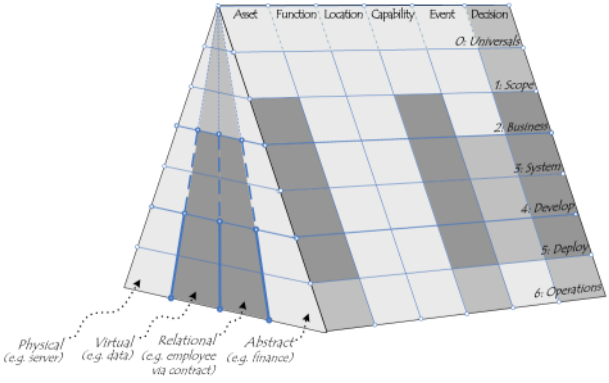


Figure 11: Architecture emphases in extended-Zachman

Finally, the *extended-Zachman framework* (Figure 11). Back in Step 1 we noted the respective scope from the project-diary – mainly about *people* and *capabilities*, though we also need to look closely at *information* (virtual assets), at people-based *events* and, especially, motivation and *decisions* in general.

For the *capabilities* or skills that we need in the architects (linked to the organisation via *relational assets*), the project-diary notes a list from an article by enterprise-architect Sally Bean:

> ***Diary:***
> *Sally Bean: 'The elusive enterprise architect' (skillsets for architects):*
> *- communicator and change-agent*
> *- visual system-thinker and modeller with fore-*

sight
- fast learner
- principled pragmatist
- incisive consultant and troubleshooter
- 'big picture' thinker

Architects also need to be consummate generalists, because they need to be able to link together every possible aspect of the enterprise, and communicate meaningfully with the specialists in each area. They usually need knowledge in breadth rather than depth – the respective specialists will handle most of the latter, although most enterprise-architects will also come from a specialist domain themselves. But even though the architect's depth of knowledge in any one skill may be quite low – sometimes just a basic understanding of key principles and technical terms – the sheer *range* of skills that need to be learned at the full enterprise-level scope may take literally decades to acquire: so despite the claims of some training-providers, true competency in architecture is *not* something that can be picked up in a single two-week workshop!

On *assets*, we've already listed the tangible *information* used in architecture: items such as glossary and thesaurus, governance-records, risks and opportunities, requirements, and many, many models. But in some ways what's even more important is the *intangible* information (a composite of virtual and relational asset, in framework terms): all those conversations and workshops and whiteboard-sessions that are so central to stakeholder-engagement and

architecture practice.

Architecture *functions* provide the interfaces through which the items affected by architecture may change. Those items would typically be information and, especially, decisions, because architecture is primarily about decision-support. The overall functions would probably be much the same in any architecture context, but the architecture *services* – and hence the effective scope of architecture – will depend on the capabilities that can be plugged into those functions. As an entry in this morning's project-diary notes:

> ***Diary:***
> *in essence architecture comes down to a single idea: things work better when they work together – role of architecture's is to ensure that things work better together over all of the respective scope*
> *(in principle, scope for enterprise-architecture is entire enterprise)*

If the only available architecture capabilities and competences are in IT, then 'enterprise-architecture' will appear to be IT-specific, and so on. Yet to extend our enterprise-architecture to the whole business and beyond, we don't need to change the architecture functions as such: we just need to extend the available capabilities – the range of skills experiences included within the architecture.

Architecture *locations* may be physical, virtual or relational, but the latter are by far the most important of these

– as in the old adage that "it's not *what* you know but *who* you know", and how and where to find those people. In the same way, we'll also need to know how and where to find the right information, the right decisions and so on. But for architecture itself we most need to know the decision-*makers* – which in practice comes down to real people, and hence the locations of those people.

The *events* for architecture itself, again, are mostly 'people-events' – relational events. (There may also be a few time-based events such as regular scheduled reviews.) These may come in many forms – emails, task-requests and many, many meetings – but the notion of an 'event' that triggers a request for action is useful here.

In principle the *decisions* assessed and acted on by architecture should again cover the entire enterprise scope, at every level from 'universals' to real-time action and review. The architecture will need some explicit means to describe the purpose, role, dependencies and jurisdictions of any or every decision in scope – the domains of the star-frame diagram being one such categorisation we might use for this. We would also need mechanisms to enable us to create a complete 'audit-trail' for any decision, all the way back to the core-'universals' for the enterprise.

For each of these entities in scope – assets, functions, locations, capabilities, events, decisions – we would need a complete, fully-maintained RACI matrix (responsible, accountable, consulted, informed) of all related stakeholders. (That's an ideal to aim for, anyway, though it might

be impossible to achieve in practice.) This would tell us who would be affected by any architectural issue, who we should engage as active stakeholders in any assessment or review, and what clashes are likely where responsibilities and accountabilities overlap.

Given the models we started with, this would complete the first pass of the assessment. It might well be useful to loop back to the start and quickly scan though again to see if any other themes or ideas come up. In any case, we should document the results in the project-diary.

Step 4: Review 'to-be' architecture against qualitative criteria

Zachman suggests that the core qualitative concern for capabilities is performance-management; to that we ought to add 'universals' such as security, health and safety, business ethics, knowledge management and the like.

For each of these we need to identify appropriate 'critical success factors' (CSFs) and metrics for key performance-indicators (KPIs). Which is not going to be easy, because the whole point of architecture is that it's about linking everything together – hence, in principle at least, its success can only measured in terms of the whole.

Many of these metrics and suchlike will depend on the industry and enterprise, so I won't attempt to list them here; but the final lists should be documented in the project-diary, as usual.

Step 5: Finalise building-blocks for architectural scope

This is a step that we will probably have to skip over here, because it's not practicable in this context.

The 'by the book' notion of Architectural Building Blocks and Solution Building Blocks, as patterns for re-use, is very useful in most parts of architecture. But architecture itself depends almost entirely on people-based capabilities – otherwise known as 'skills' – and the concept of 're-use' doesn't work in the same way with people as it does with software or machines or other physical 'things'. One important reason is that people embody skills in very different ways to those in which we build capabilities into machines or software; another is that most of the architect's skills are in the 'unorder' domains, which are naturally not amenable to simple re-use. Either way, probably best for now to document it as 'not applicable', and move on.

Step 6: Conduct checkpoint-review for stakeholders

This again would be simpler than usual, because the primary stakeholders are us. Perhaps the most important point would be to review the list of requirements to date, recorded in the project-diary as follows:

> **Diary**:
> *- suitable reference-models, standards, techniques for architecture*
> *- central role of sensemaking; role of architecture versus design*
> *- checklists and other 'rotations' for the assessment toolkit*

- fluency in sensemaking to select checklists, views and 'rotations'
- fluency in identifying recursion, reflexion and similar patterns
- fluency in strategic assessment (Purpose phase)
- fluency in 'soft-skills' / people-skills (People phase)
- fluency in analysis and modelling skills (Preparation phase)
- familiarity and practice with architecture methodology (Process)
- appropriate performance-metrics for architecture (Performance)
- assets: workspace, computer, whiteboard etc (physical); information-sources and -stores (virtual); access to people (relational): executive support for architecture (aspirational)
- functions: processes for architecture, including governance, quality, process-improvement, engagement and delivery
- locations: strong social-networks across and beyond organisation
- capabilities: generalist-level skills for all competencies across enterprise scope; specialist-level skills in architecture itself
- events: contexts and interface-specs for architecture-events
- decisions: categories for decision-types; facility for dependency / validation 'audit-trails' for any decision (or other entity)

- RACI matrices associated with all architectural entities
- appropriate set of metrics (KPIs) etc and success-factors

If necessary, we could loop back to step 3 above to re-assess our architecture-models and the list of requirements we've derived from them.

Once we're comfortable with that, we're ready to move on to do the same kind of assessment with our current or 'as-is' context for 'the architecture of architecture'. Before we do that, though, we need to do the first assessment for our example-project.

Example project: 'as-is' assessment for the bank

What should we do to assess the organisation's architecture of respect? That's the challenge here…

We have a couple of workshops planned for this day: one for the executive, one for frontline workers. That'll be important, because those are likely to be our main information-sources for this project – there's not much else to be had. But let's at least start off doing it 'by the book', and see what happens from there. We're using the 'as-is' as the primary time-horizon, because the overall aim is to recreate in the future the respect that existed in the past

but does not exist now – hence the place where we need to focus most of our attention is on what's changed between past and present.

Step 1: Develop baseline architecture for 'as-is' context

This is short and sweet – or not-sweet, rather, because we have no information available from which to derive that baseline.

Step 2: Select reference-models, views and viewpoints

There are no standard models for 'respect' as such, but a decision-modelling standard such as the Business Motivation Model could be useful here, especially if linked to higher-level models such as the 'Vision Role Mission Goal' framework. For consistency, we'll also make what use we can of those four main base-frameworks:

- five-domain context-space mapping
- five systems-theory principles
- five-elements lifecycle
- extended-Zachman framework

Ideally we would model for a wide range of viewpoints, both inside and outside the organisation. Realistically, though, we'll have to make do with whatever viewpoints we can derive from that pair of workshops, together with any information we can glean from other sources about the views of 'outsiders'.

Step 3: Create and update 'as-is' architecture models

The first stage of the assessment is the two workshops: we'll then develop suitable models from the results.

Workshop for executive

This is organised as a half-day offsite session for the executive and other senior staff. Just under thirty people in all, of whom barely a third arrive on time – the remainder drift in over the next half-hour or so, without apology. Many of them ignore the no-phones rule for the session – including the CEO, who seems to be up and down like a jack-rabbit following one phone-call after another. The CIO even keeps her laptop open throughout the session, pounding away on an endless stream of 'urgent' emails.

We use the five-element model as a central focus for discussion. It soon becomes clear that, like many business organisations, they're strong on Preparation and Process – planning and production – but not so strong on Performance – completions, follow-up and strategic use of metrics – and frankly weak on the Purpose and People domains. The nearest equivalent to strategy, for example, seems to consist of receiving a list of quarterly financial targets from global headquarters and then hacking out a quick plan that might deliver the required results – which could hardly be called a strategy at all. And the 'people' aspects of overall planning were no better: the CIO, for example, was visibly overloaded from the strain of trying to complete the integration of the former banks' IT-systems, but she was doing so without any real support from the rest

of the business.

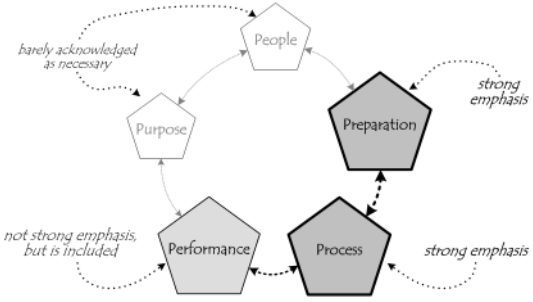


Figure 12: Five-elements model for bank

The theme of 'respect' provides some interesting views into the overall context. Whilst it's clear that professional respect between them is more than adequate, that's not reflected in their mutual *actions*: people talking over each other, others talking amongst themselves whilst someone is supposedly presenting to the whole group, and, of course, the ubiquitous Blackberrys and excuse-me-I-just-have-to-take-this-call. There are a few displays of oversized egos, of course, but overall there's not as much political infighting as we've seen in other organisations, which ought to be a good sign – yet it seems extraordinarily difficult to get them to work together as a *team*. In that sense, respect *is* a serious problem here, right down to the way the senior management work with each other – or *don't* work with each other, more accurately.

What's not clear is what to do about all of this. But we shouldn't concern ourselves about that at this point

anyway, because, as noted in the project-diary:

> ***Project-diary:***
> *must exclude any consideration of 'solutions' during the assessment phase – document any ideas, but don't discuss!*

Instead, we simply take note of whatever information comes up, and hold back on any assessment until later.

Workshop for operations staff

This second workshop is a much larger affair, several hundred staff happily crammed into a local theatre. For this we've joined with another team who have been running a more conventional organisational-development programme involving music, group-work and so on. It's also all in the local language, so we would otherwise only have been able to work through a translator – whereas here we're able to slip our questions into the overall mix, and note what comes up as a result.

There are staff here from almost every area of bank operations: tellers from main branches and in-store franchises, back-office staff, call-centre workers, a few technical-support people. What becomes clear straight away, if masked by the laughter brought on by the skilled presenter, is that there are huge conflicts here, many of them caused by conflicting goals set by management for each group and business-unit: for example, one team's bonus is based on how much they cross-sell credit-cards to clients, whilst

another team's is based on how much they *reduce* clients' over-credit. And *all* frontline staff – both customer-facing and call-centre – are now bearing the brunt of customers' ire at the bank's response to the current worldwide 'credit crunch', suddenly switching from an apparent policy of throwing credit-cards around like confetti, to endlessly haranguing every cardholder to pull back on their credit. At the executive level the figures may look good, for now at least, but at the front-line *no-one* is happy… and that definitely does *not* augur well for the future.

Again, no 'solutions' at present; yet it's noticeable that the simple fact of being *heard* seems to have made a real difference to the way these operations-folks now view their work.

Follow-on assessment

We start with *context-space mapping*.

A quick summary would suggest that – as is again common with many organisations – there's been an assumption somewhere that everything fits within the 'ordered' domains, with little allowance for the reality of 'unorder'. Hence the too-Simple attempt to 'take control', trying to force the market to change its behaviours to suit the organisation's necessarily changed policies – and hence also an inevitable fallback to a market-context that has become 'chaotic' in the wrong sense of the word. Much the same applies to the results of endlessly-repeated cost-cutting within the organisation itself: yes, it's 'lean and mean', but there are now no reserves left to deal with the stress

of change, and the strains are beginning to show, all the way up to the executive – not least in the CIO's increasing difficulties in keeping up with her insane workload.

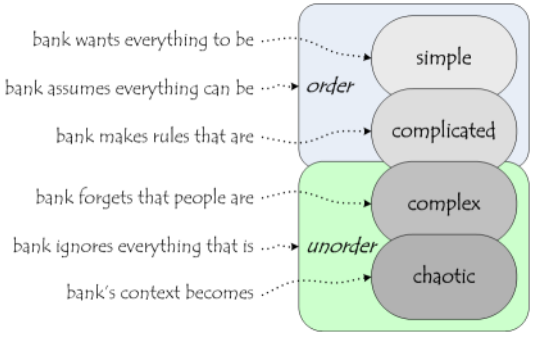


Figure 13: Bank example: collapse from control to chaos

Next, the *systems-theory principles*.

It's clear that both *recursion* and *reflexion* are in play here, because the overall 'respect'-problems and stress-symptoms are all too evident throughout the organisation and enterprise (recursion), and can be seen in almost any point within it and from outside it (reflexion).

There's evidence of failure to apply *rotation*, a consistent overview from every perspective – instead, each person seems to view the enterprise solely from their own standpoint, without much sense of the whole, or expressed feeling of belonging to a whole.

And there's also not much evidence of *reciprocation*, in that the organisation seems to have a very one-sided, even

self-centric view of its relationship with its market: in the CEO's eyes at least, 'shareholder-value' comes first, with clients' needs a very distant second. The market-model suggests that this would lead to a very destructive *resonance* feedback-loop that would lead to spiralling damage to the bank's reputation – exactly as reported and as we've also seen for ourselves in those two workshops.

On to the *five-elements lifecycle* review.

This we can summarise direct from the executive workshop: the main emphasis is on Preparation and Process phases, with some on whole-system use of Performance, but nothing like enough of Purpose or People to provide much-needed balance. Seriously lopsided, in other words, if not unusually so for a commercial organisation – but the longer-term impacts of that imbalance are now starting to show.

And architectural entities in scope, in *extended-Zachman* terms.

Almost all of this is about people, values and feelings – or, in architectural terms, relational-assets (links with real people), 'universals' (values) and the 'audit-trails' of items and interactions (particularly feelings) that link them together. Architecturally, what's missing here is an explicit description of values to which everything can be anchored – the *vision*, or, as one of our other clients put it, "the totem-pole to unite the tribes". Just what that would be is far from clear at present: all that *is* clear is that 'shareholder-value' isn't it, if only because that's of no interest to

the bank's clients from whom that supposed 'shareholder-value' would ultimately be derived.

In the project-diary, we also carried forward two notes from the previous phase:

> ***Project-diary:***
> *- for phase B/C: will need to identify policies, rules and regulations that would apply to detail-assessment and implementation*
> *- for phase B (as-is): use POSIWID to assess implied-'purpose' of current systems*

Policies, rules and regulations may not apply until we start some kind of implementation – which may not happen in this case, and may not be our responsibility anyway. But we note an interesting clash of cultures: this country's culture places a strong emphasis on the family, the collective, whereas that of the parent-company builds everything around the individual – and most of the bank's metrics and bonus-structures reflect the values of the parent, not this place. That in itself might be a source of problems that need further exploration.

POSIWID is an acronym coined by the cyberneticist Stafford Beer, to describe the relationship between purpose and practice: "the purpose of a system is [expressed in] what it does". In effect, the bank is a 'system' whose current effective purpose is to create the problems that we'd seen in that assessment. No-one *designed* it to create those problems, of course – though 'design' is actually a key

source of the problems here, because these issues can only be resolved not by tackling them piecemeal, but by tackling them as a whole, *as* a single unified system. That doesn't mean that we should try to do everything at once – that's impossibly disruptive, and it never works anyway. What it *does* mean is that we need to think architecturally, in terms of the architecture of the *enterprise*, not just the organisation. And one key item that's most notable by its absence, perhaps, is the lack of any meaningful enterprise-scope vision – so that might well be a good place to start work.

But that's a 'solution' – and all would-be solutions should be shelved until we get to the appropriate point, which is not here. For now, though, we've finished this part of the assessment: time to move on.

Step 4: Review 'as-is' architecture against qualitative criteria

To the CEO, it seems, almost the only thing that matters is the quarterly figures; but that won't work well enough in this context – not least because money won't buy respect. The whole focus of this exercise is qualitative, not quantitative – but it's still far from clear yet as to which qualities we'll need to focus on. Something to note in the project-diary, but otherwise just keep going.

Step 5: Finalise building-blocks for architectural scope

If we were to do this 'by the book', we should look for re-usable building-blocks at this point. But once again that building-blocks concept doesn't quite make sense here,

because almost all of this is about people rather than 'things'. Again, probably something best left to review in the next phase.

Step 6: Conduct checkpoint-review for stakeholders

We finished all of the above review within about half an hour after the second workshop ended. Our client – the bank's change-manager – was still helping the main presenter to pack up at that point, so we're able to discuss the assessment so far. He's very pleased with what sees, so we get his go-ahead for the next phase – the 'to-be' assessment.

Application

- How do you do architectural-assessment at present? What techniques, toolsets and methods do you use?
- How do you avoid 'premature fixation' on a solution during the assessment phase of architecture?
- How would you assess the architecture of architecture itself?
- If you're already doing enterprise-architecture, what domains of the enterprise does this architecture cover? If it does not describe all elements of the organisation and key elements of the extended-enterprise, what assets, functions, locations, capabilities, events and decisions would be needed for it to expand outward

to cover that full scope? What are or would be the consequences to the organisation if it does *not* fully cover that scope?

- How would you use architecture to tackle a big-picture business-issue such as "to enhance respect in the marketplace"? (Or – as in the case of a real business metric used by one of our government clients – "to increase the number of days between bad headlines in the newspaper"?) Where would you start? What planning and governance would you need for the assessment itself?

- What do you see if you apply POSIWID to your own organisation's context and scope? If "the purpose of the system is what it does", what would you change in that purpose, and why?

More on context-space mapping

To make sense with context-space mapping, we first need to go right back to first-principles: the core concept of context-space.

Before any notion of order or unorder, or even of dis-order, there is simply 'the everything': everything and nothing, all one, with that 'everything-and-nothing' linked to everything-and-nothing else, in a place-that-is-no-place that incorporates within itself every possibility. It's not 'chaos' – it simply *is*.

There are all manner of names for this 'active no-thing-ness': Lao Tse called it 'the Tao', for example, whilst the ancient Greeks described it as 'the Void'. For the more business-oriented purpose of enterprise-architects, though, we'll need to constrain the scope of this 'the everything' somewhat, and we'll also need a more 'business-like' label. So let's call it *context-space* – the holographic, bounded-yet-unbounded space that contains every possibility within the chosen context.

Figure 42: Context-space

Elsewhere in this book I've split this context-space into

problem-space – the context in which things happen – and *solution-space* – the space in which we decide what to do in relation to what's happening. But ultimately there's just the context: "the only true model of a system is the system itself".

Yet to make sense of anything, we need to impose some kind of structure. One place to start would be to filter 'the everything' in terms of its variability. Perceived-repeatability is one obvious example of a variability that we might find useful, but there are of course many others.

At the start, this gives us a finely-graded spectrum of variability across the context (Figure 43).

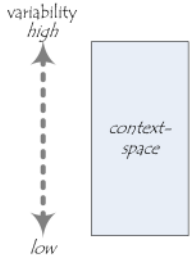


Figure 43: Variability in context-space

Interestingly, though, most human sensory-perception does not work well with smooth gradations: it works much better with distinct boundaries. Hence most sensemaking will usually attempt to place some kind of ordered structure upon what may initially seem like unbounded chaos, to act as a filter that can help us separate 'signal' – that which

we're interested in – from 'noise' – that which is not of apparent interest at present.

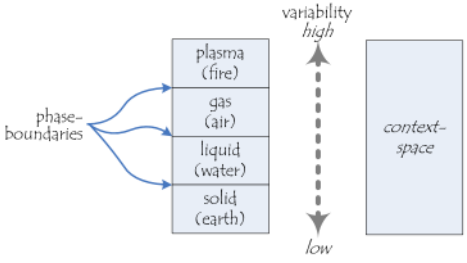


Figure 44: Phases of variability in context-space

For example, when we look at the physical world of matter and material, we can see both of these processes in action, even within matter itself. There is a fairly smooth gradation of variability, primarily linked to temperature; yet there are also explicit 'phase-boundaries' where the internal relationships of matter undergo fundamental changes. Significant amounts of energy ('latent heat') can be absorbed or released in the 'phase-transitions' between these modes. In effect, these will present as four distinct states of matter, traditionally described as Earth, Water, Air and Fire, for which the respective scientific terms are Solid, Liquid, Gas and Plasma (Figure 45).

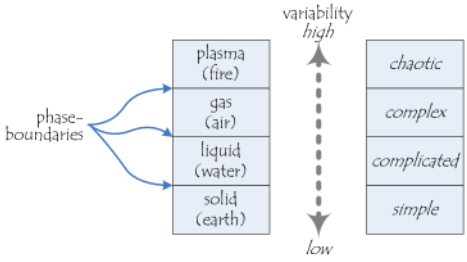


Figure 45: Phases as domains of context-space

When we look at the *internal* structures of matter within each of these states, we would typically describe the respective structural relationships as Simple, Complicated, Complex and Chaotic, as phases or domains within the context-space of matter. This type of categorisation along a single axis represents a simple first-order map of that context-space – hence context-space mapping.

Much the same applies to just about any other view into that overall context-space. If we take almost any type of gradation, we will be able to identify distinct phase-boundaries that can be used to partition the context-space into distinct regions along that axis: one such example is the nominal split of the visible-light spectrum into Red, Orange, Yellow, Green, Blue, Indigo and Violet. But perhaps the most useful split of all for enterprise-architecture and business-architecture is along an axis of repeatability, dividing the inherent uncertainty of context-space into regions that, in parallel with those states of matter, we could describe respectively as Simple, Complicated, Complex and

Chaotic.

On the surface at least, this brings us into a similar conceptual space that of the Cynefin framework – though we've arrived there via what is, in very literal sense, a fundamentally-different route. And here we can also see:

- how and why we've arrived at those particular categorisations
- how and why to use any specific axis for such categorisation
- what the boundaries between the 'domains' in the categorisation will look like
- how, why and when the nominally-Simple boundaries between categories may move (Complicated), blur (Complex) or fragment (Chaotic).

This provides a layered, recursive richness that is largely absent in the standard Cynefin frame. It also provides a means to link right across every possible view into context-space, rather than solely a specific set of interventions that focus primarily on a set of views into the Complex domain.

A first-order (single-axis) context-space map – such as the Simple-to-Chaotic 'stack' – is not all that much use in practice. To make it more useful, we'll need to add other axes as filters for sensemaking, to enable relevant information to fall out of the respective comparison. And we make it more useful again by selecting a related set of

axes to provide a multi-dimensional base-map upon which other filters can be placed.

> Two-dimensional base-maps are the easiest to work with, for obvious reasons, but three or more dimensions are entirely feasible – the tetradian (see Figure 32 and Figure 33 in chapter *Day 8: Putting it into practice*) is one example of a four-dimensional frame compressed into three-dimensions for use as a base-map.

To do this, we choose axes which force the domains of the original single-axis spectrum into relations of opposition and similarity with each other. For example, if we use 'levels of abstraction' as the core axis, and overlay that with timescale in one direction and a 'value-versus-truth' spectrum in the other, we arrive at the following base-map and its 'cross-map' of interpretive text-overlays:

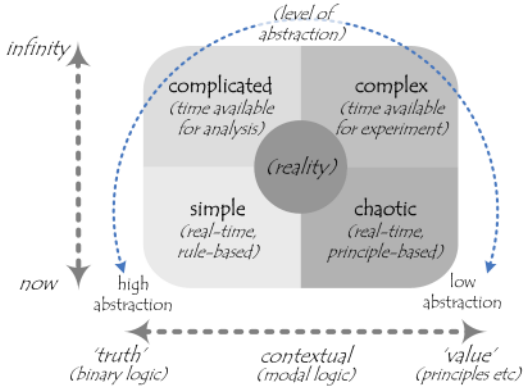


Figure 46: Context-space: abstraction, interpretation, timescale

Here Chaotic and Simple are opposites in their interpretations, but similar in terms of timescale; Chaotic and Complex are similar in their means of interpretation, but opposites in terms of timescale; Simple and Complex, and Complicated and Chaotic, oppose each other on both axes; yet all domains are related in terms of layers of abstraction. The central region ('reality') is essentially a reminder that the domains represent related yet arbitrary views into what is actually the total 'hologram' of context-space – everything else is actually an abstraction from the real.

We then layer this recursively to apply to the nominal boundaries between each of the domains, so that these too may be considered to be fixed, movable, porous or fragmented or transient. An axis based on a binary 'true-or-false' categorisation (in other words, a Simple boundary) will split the context-space into two domains along that

axis. If both overlay-axes have Simple categorisations (or movable two-part categorisations, in Complicated style), the overall context-space is split into four regions – which aligns well with the 'matter'-type categorisation of Simple, Complicated, Complex and Chaotic. Likewise a smooth gradation along both axes pushes the context-space into four regions with Complex or even Chaotic boundaries between them.

Because of this, a four-region base-map is likely to be the most common two-dimensional type: for example, the standard Cynefin frame is often shown (or, technically, misused) as paired with two-axis overlays. But other layouts are possible and sometimes useful: for example, a pair of tri-value axes would typically be used to align an eight- or nine-domain primary axis, such as seven-colour plus infra-red and ultra-violet.

The result is a consistent structure for base-maps that are both bounded *and* not-bounded, and that describe the whole of a context-space by structured views into that context-space that also acknowledge that the context-space itself has no actual structure.

> Hence perhaps important to note that whilst Cynefin may sometimes be shown with two-axis overlays – and appropriately used as such for context-space mapping – it is not in itself solely a two-axis matrix. And useful though it is, it's merely one instantiation of a generic class of context-space base-maps that has been

around and in general use for decades, if not centuries. Once again, it's important to remember that Cynefin and context-space-mapping *are* different, and do have different roles and functions in the overall process of sensemaking and decision-making.

Cynefin cross-map

The standard 'Cynefin diagram' on Wikipedia provides one such example of a cross-map for sensemaking:

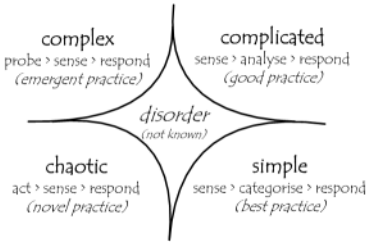


Figure 47: Standard Cynefin cross-map

This shows two cross-maps overlaid on top of the 'Simple, Complicated, Complex, Chaotic, Disorder' categorisation and layout of the Cynefin base-frame:

- typical tactics to use where the respective cause-effect conditions apply (such as 'probe > sense > respond' for 'Complex' causality)

- common terms for the overall practice in which such tactics are used

  Note, though, the crucial difference between that simple cross-map diagram and the processes of sensemaking which make use of that cross-map. In the Cynefin framework itself, a variety of diagrams of this type are further cross-linked to "research into complex adaptive systems theory, cognitive science, anthropology and narrative patterns". The standard framework "proposes new approaches to communication, decision-making, policy-making and knowledge management in complex social environments". 'Cynefin' is really that whole framework, rather than the diagram or categorisation or layout – hence the real importance of the various pedantic-seeming clarifications that I've placed throughout this book.

Note that the tactics shown in this cross-map are only *typical*, not mandated – if only because that might be too Simple an approach! For example, relying solely on the 'act > sense > respond' tactic that's listed for the Chaotic domain implies that the *only* appropriate response to natural chaos is to 'run away' to some other domain – either by a Simple tactic of 'taking control', or the more Complex tactic of applying abductive-reasoning to whatever information arose in those brief moments of chaos.

In many cases – especially in architectural assessment – we would be better served not by 'running away', but by intentionally turning *towards* the natural panic that occurs in the Chaotic domain, so as to give the not-so-random information that we need more time and space in which to arise. The best guide in each domain is to cross-map with the respective decision-types: rules in the Simple, algorithms in the Complicated, heuristics or guidelines in the Complex, and principles in the Chaotic.

Jungian-type base-map ('embodied best-practice')

This cross-map draws on Jungian concepts – a two-axis matrix of internal versus external, and 'truth' versus 'value' – yet it also provides a further cross-map with that Cynefin cross-map's types of practice. This is particularly useful for sensemaking in skills-development and in implementation of innovation.

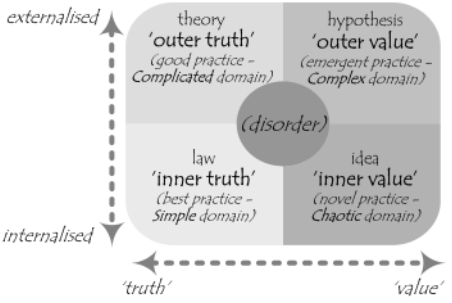


Figure 48: Embodied best-practice

This also cross-maps to the earlier example (Figure 46, above), in terms of available time. Initial 'useful ideas' need sufficient time to develop hypotheses for reflection, experiment and test, and further refinement into theory, which will usually anchor it into a conventional linear (true/false) cause-effect logic. This then becomes enshrined as 'law' or 'best practice', or (in skills-development) is literally embodied through repeated practice, such that the action can become an automatic response available for use in real-time skills-implementation.

In effect, there is a 'counter-clockwise' pattern or pathway here traversing through the conceptual space of the cross-map, a cycle of continuous innovation. (As will be seen later, the well-known PDCA practice-improvement cycle actually goes in almost the opposite direction.) As can also be seen in Figure 8 (back in chapter *Day 3: What's going on?*), one of the real dangers that this highlights is that the 'law' domain is often regarded as a final destination, an apparent guarantee of certainty. In reality this seeming 'finality' of 'law' is actually spurious and misleading – as is indicated by the fact that 'law' is a single region of the conceptual-space here, not the whole of the space. Wherever this notion becomes dominant, the boundary between 'law' and 'idea' can become near-absolute, preventing further innovation even when necessary. In such cases, innovation may become possible only via a transit through the region of 'disorder' – which in a business context can sometimes be disruptive in just about every possible sense…

Repeatability and 'truth'

A straightforward cross-map, this one, and very useful in many aspects of enterprise-architecture and the like. This is one example where it might make more sense to show the domains as a vertical stack (see Figure 45 earlier).

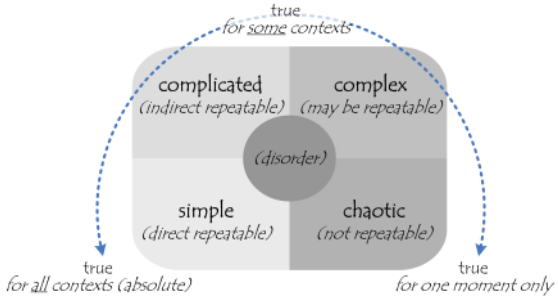


Figure 49: Repeatability and 'truth'

The real-world is 'disorder': everything else is an abstraction. The Chaotic domain is the simplest abstraction, in that it asserts that, in principle, everything is context-dependent and nothing is repeatable. The Simple domain represents the opposite extreme, in that it asserts that there is 'absolute truth' and perfect repeatability – with a concomitant tendency to complain that the real-world is being somehow 'unfair' or 'wrong' if it does not conform to the expected 'truth'. The Complex and Complicated domains sit somewhere on the spectrum between these two extremes, with the Complicated domain preferring stronger abstractions, and the Complex domain accepting

that reality isn't quite that simple.

Marketing versus sales

A useful cross-map that explores the perennial clash between Marketing and Sales. This draws on that dimension of timescale, from infinite to immediate, and on a less commonly-used yet perhaps more important dimension: the concept of ownership, across a spectrum from *possession* – the default view in modern societies – to *responsibility* – which is actually more common in practice within organisations themselves.

Figure 50: Marketing versus sales

Once again, the reality of the market is that it is 'disorder': any other view of it is an abstraction.

The most Simple market is a monopoly. You alone set the rules, and others (especially the 'consumers') have no

choice but to buy according to your rules. In an all-too-literal sense, you *possess* that portion of the market, and hence also that portion of people's lives. Much of it is about trying to control what people do, often in a very physical sense: people are treated as objects or subjects rather than *as* people.

It makes marketing very simple – in fact ideally there is no need for any 'marketing' as such – but there are two very real dangers. One is that it's an extreme abstraction of reality, and if reality moves away from alignment with that purported 'truth', the market can sometimes vanish overnight – as happens quite often on the internet, for example. The other is that monopolies often breed deep-seated resentment, and if the monopoly cannot be bypassed, the resentment may explode elsewhere – as happened with British monopolies on salt, fabrics and many other essential items in colonial India. We see much the same in lesser form with Microsoft's current dominance of the operating-system and office-software markets – contexts where a 'natural monopoly' will tend to occur simply because of the need for standardised information interchange. So whilst possession may *seem* like the best possible strategy, the long-term consequences can be much more severe than they look.

Most conventional marketing sits firmly in the Complicated domain: crunch the numbers, map the trends, analyse every-which-way to find out how to make the market predictable. People tend to be regarded as units of information, a datapoint within the statistics, rather than as

individual people; in fact it's very much about information, the conceptual dimension, and often also about trying to 'control' what people think about a product or service. (Trying to determine what people feel pushes the emphasis more towards the Complex domain, whilst the common notion here of 'taking control' of a market pushes the emphasis the other way, towards the Simple domain.) Note also the cross-map with timescale: marketing may occur *before* or *after* but not *at* the exact moment of sale.

We move into the Complex domain of marketing by regarding people more *as* people rather than as 'consumers'. Complexity demands much more acceptance of human factors, of 'wicked problems', and also a weakening of the separation between 'us' ('producers') and 'them' ('consumers') – as can be seen in the success of Amazon's customer-driven ranking as a marketing strategy, in some forms of crowdsourcing, and also in Agile-type development where the customer is also part of the development-team. The central theme is about relationships, which, although still 'abstract' in terms of timescale, may in effect extend and push the boundary of this domain quite a long way towards real-time, into what would otherwise be Chaotic space.

Yet by definition, Sales themselves reside in the Chaotic domain, because every decision to buy or not-buy is a quantum-event, a 'market-of-one'. The ultimate drivers for all such decisions are values-based, not 'rational' or 'truth'-based, which means – as just about any good salesperson would tell us – that the focus here is on *aspirations*. Given that sales deals with real-time events, we're somewhat

forced into the principles-versus-rules spectrum: online sales will go toward the rule-based end of the spectrum, because that's all that IT systems can handle, but real salespeople working face-to-face with real clients or customers (not 'consumers' here) will recognise key principles such as the need to listen – and also to know when to stop talking, so as to allow space for the decision to take place.

This cross-map also shows us that, by definition, the conventional approaches to sales and marketing are diametrically opposed by the nature of what they do and how they work; but we can bridge that gap somewhat via either the Complex domain of emergent marketing, or by the Simple domain of IT-based sales (supported, again, by cross-links to the Complex domain to remove the risk and resentment around perceived monopolies). Which approach is 'best' in each case will depend on the context – which this cross-map, and others, will again help us to identify.

Plan / do / check / act

Another very useful cross-map that helps to clarify what's actually happening within the PDCA improvement-cycle, and is also a good illustration of the *dynamics* in context-space mapping.

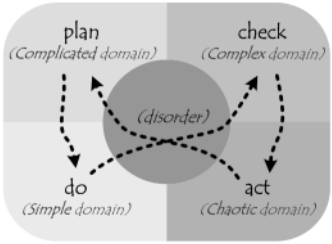


Figure 51: The Plan / Do / Check / Act cycle

The cycle starts with Plan. This is primarily about information, and takes place before real-time contact, both of which tend to place it in the Complicated domain.

The aim of the Plan is to create rules that are Simple enough to apply in real-time when we Do the actual work. Although 'work' can take many forms, it still needs to be made concrete in some way in the real world, which in effect places an emphasis on the physical dimension.

The work is not abstract: it happens in the real world, in real-time – in other words, in terms of the star-diagram frame, a transit through 'Disorder'.

On completion, we move back out of real-time to reflect on the difference between what we'd intended (Plan and Do), what actually happened (a transit through Disorder – the difference between abstract intent and concrete reality), and what we can do about it (Check, leading to Act). Learnings need to be both personal and collective, which places us on the 'values' side of the 'truth'/'value' spectrum; long-term experience indicates that such learning takes

place in a social or relational context, away from the action, through tactics such as After Action Reviews – all of which indicates that this part of the cycle situates in the Complex domain.

The outcome of the Check phase is a set of guidelines for revised future action, on which we need to Act so as to embody the required changes in personal awareness and action, via a personal review of the underlying principles of the context and how they apply to that specific individual. To change how we work also requires that we face the personal challenges implied by any kind of change, so it's also about personal aspirations and personal responsibility, in the sense of 'response-ability' – the ability to choose appropriate responses to the context in real-time action. Ultimately all of this is unique to the individual, a 'market-of-one' – and hence places this phase of the PDCA cycle in the Chaotic domain.

We then wait for an appropriate new real-world context – in other words, another transit through 'Disorder' – to start the cycle again with a new Plan.

This cycle is also echoed in the problem-solving method first proposed by the Hungarian mathematician George Polya in his 1945 classic *How To Solve It.* The steps in his cycle are: Understand the problem; Devise a plan; Carry out the plan; Review and extend – which is the same as PDCA, but starting one step earlier, where PDCA's 'Act' includes a re-understanding of the problem.

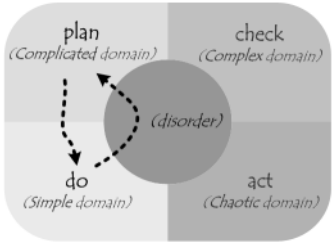


Figure 52: Failure-path – Plan / Do loop

There are several ways in which the cycle can fail. One is that an obsessive production-oriented context skews the path through Disorder, to give a tighter loop of Plan / Do / [Disorder] / Plan (see Figure 52). This cuts out Check and Act – which may seem unnecessary in the short-term, but is probably disastrous in the medium- to longer-term, since it assumes that the rules created by the plan will always apply. Not so much Simple as dangerously simplistic…

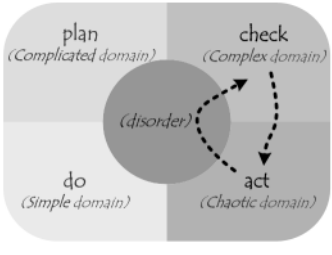


Figure 53: Failure-path – Check / Act loop

Another type of failure occurs when extreme self-doubt skews the other return-path through Disorder, to give a

probably even-tighter loop of Check / Act / [Disorder] / Check (see Figure 53). In effect, this is a kind of personalised version of 'analysis-paralysis' – much may be learned, but nothing is actually done!

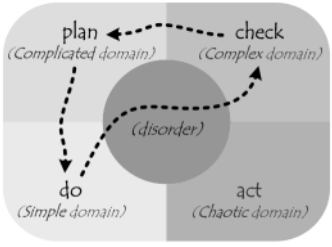


Figure 54: Failure-path – Plan / Do / Check loop

Yet another failure-loop is Plan / Do / [Disorder] / Check / Plan (see Figure 54), in which the review takes place, but pressure of work forces a return to the Plan phase before any actual change can be embedded in personal action. This is perhaps the least effective form of 'process-improvement', but seems depressingly common in real-world business-practice.

ISO-9000

A fairly straightforward cross-map to something that's usually presented as a vertical stack but actually makes more sense in a star-diagram layout (Figure 55):

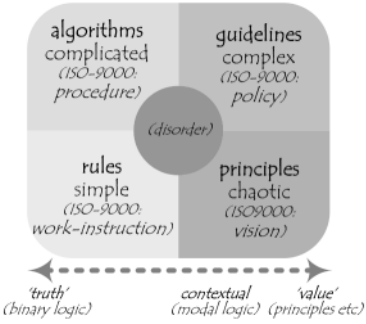


Figure 55: ISO-9000 cross-map

A *work-instruction* defines Simple rules that apply to a specific context. In Zachman-framework terms, it provides the row-4 or row-5 detail-level What, How, and Who that apply at a specific When-event, with Where usually defined in more generic terms (such as any location that uses a specific machine). The underlying Why is usually not specified.

When anything significant needs to change – for example, a new version of software, or a new machine – we move 'upward' to the *procedure* to define new work-instructions for the changed context. This accepts that the world is more Complicated than can be described in simple rules, yet is still assumed to be predictable. The procedure specifies the Who in terms of responsibilities, and also far more of the underlying Why – the row-3 'logical' layer, in Zachman terms.

When the procedure's guiding reasons and responsibilities

need to change, we move upward again to *policy*. This provides guidance in a more Complex world of modal-logic: in requirements-modelling terms, a more fluid 'should' or 'could' rather than the imperative 'shall'. The policy describes the Why for dependent procedures – the row-2 'conceptual' layer, in Zachman terms (though 'relational' might be a more accurate term here, as we'll see from other cross-maps).

When the 'world' of the context changes to the extent that the fundamental assumptions of current policy can no longer apply, we turn to *vision*. This is a core set of statements about principles and values that in effect define what the enterprise is. Because this vision should never change, it provides a stable anchor in any Chaotic context – in Zachman terms, either the row-1 'contextual' or row-0 'universals' layers (though again 'aspirational' might be a more useful term here).

Note that in some ways this cross-map is the exact opposite of the 'Repeatability and 'truth" cross-map earlier: there, the purported 'universality' of a given 'truth' increases as we move from Chaotic to Simple, whereas here the values become more general and broader in scope as we move from Simple to Chaotic.

Skill-levels

This cross-map links to a well-known and very useful heuristic on the amount of time that it takes to develop specific levels of skill (Figure 56):

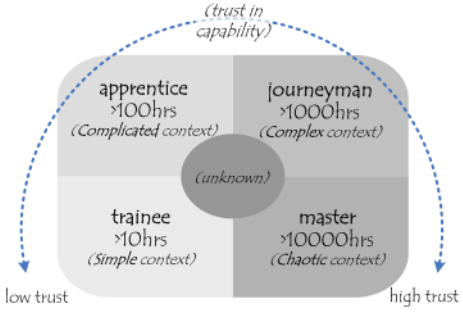


Figure 56: Skill-levels

The 'trust in capability' spectrum here is actually an inverse of the amount of supervision needed both to compensate for lack of skill and to shield the person from the consequences of real-world complexity and chaos in that context.

A *trainee* can be 'let loose' on Simple tasks after about 10 hours or so of practice (a 1-2 day training-course).

An *apprentice* will begin to be able to tackle more Complicated tasks after about 100 hours of practice (2-4 weeks); most of those tasks, however, will still need to be insulated from real-world complexity.

A *journeyman* will begin to be able to tackle more Complex tasks that include inherent uncertainties after some 1000 hours of practice (6 months full-time experience). Typical uncertainties include variability of materials, slippage of schedules, and, above all, people. Traditionally there is an intermediate point within the 1000-10000 hour range at which the person is expected to go out on their own with

only minimal mentoring: in education this the completion of the bachelor's degree, whilst in a traditional technical training this is the point at which the apprentice becomes qualified as a literal 'journeyman' or 'day-paid worker'.

A trainee should reach a *master* level after about 10,000 hours (5 years) of practice - the traditional point at which a journeyman was expected to produce a 'master-piece' to demonstrate their literal 'mastery' in handling the Chaotic nature of the real-world. This is also still the typical duration of a university education from freshman to completion of masters' degree.

Skill should continue to develop thereafter, supported by the peer-group. Building-architects, for example, often come into their prime only in their fifties or later: it really does take that long to assimilate and embody all of the vast range of information and experiences that are needed to do the work well. Hence there is yet another heuristic level of 100,000 hours or so (more than 50 years) – which is probably the amount of experience needed to cope with true Disorder.

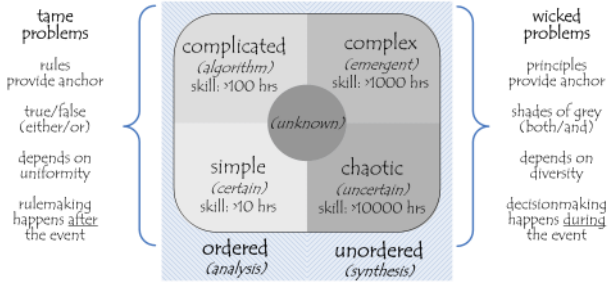


Figure 57: Skills, problem-types and decision-making

Another skills cross-map (Figure 57) shows why this isn't as straightforward as a simple linear stack. In the early stages of skills-development we in effect pretend that each context is predictable, controllable, reducible to some kind of ordered system; but at some point in the apprenticeship there's a crucial stage at which we demonstrate that the world is inherently uncertain, inherently 'unordered'. In the real world we can learn to *direct* what happens, yet it can never actually be *controlled* – a distinction that is sometimes subtle but extremely important, and actually marks the transition to true skill. As indicated in the cross-map above, there are *fundamental* differences in worldview on either side of that transition.

Automated versus manual processes

This final cross-map is a logical corollary from the skills-maps above, although it also has cross-links with the 'Asset-types' map (see Figure 32, in chapter *Day 8: Putting*

it into practice). It's reasonably straightforward, but it also has extremely important implications for systems-design.

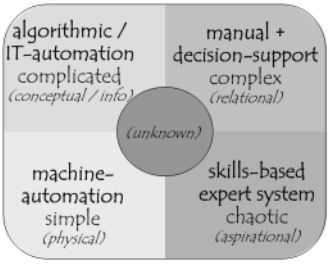


Figure 58: Automated versus manual processes

Physical machines follow Simple rules – the 'laws of physics' and the like. The Victorians in particular did brilliant work exploring what can be done with mechanical ingenuity – such as Babbage's 'difference engine', or, earlier, Harrison's chronometer – but in the end there are real limits to what can be done with unassisted machines.

Once we introduce real-time information-processing, algorithmic automation becomes possible, capable of handling a much more Complicated world. Yet here too there are real limits – most of which become all too evident when system-designers make the mistake of thinking that 'complexity' is solely a synonym for 'very complicated'.

As with skills-development, there is a crucial crossover-point at which we have to accept that the world is not entirely repeatable, and that it does include inherent uncertainties. One of the most important breakthroughs in IT-

based systems here has been the shift to heuristic pattern-recognition – yet there are real dangers, especially in military robotics, that system-designers will delude themselves into thinking that this is as predictable as it is for the Complicated contexts. Instead, to work with the interweaving relational interdependencies of this Complex domain – especially the real complexities of relations between real people – the best use of automation here is to provide decision-support for human decision-making.

In a true Chaotic context, by definition there is little or nothing that a rule-based system can work with, since – again by definition – there are no perceivable cause-effect relationships, and hence no perceivable rules. The only viable option here is a true expert skills-based system, embodied in a real person rather than an IT-based 'system', using principles and aspirations to guide real-time decision-making. One essential point is that there is no way to determine beforehand what any decision will be, and hence how decisions are made. Although there indeed a very small number of IT-based systems that operate in this kind of 'world' – such as those based on 'genetic-programming' concepts – we have no real certainty at the detail-level as to how they actually work!

Note that most – perhaps all – real-world contexts include a mix of all of these domains. This is why any real-world system must provide appropriate procedures for escalation and de-escalation: moving 'upward' from Simple to Complex to handle inherent-uncertainty via human skills, and 'downward' from Complex to Simple to make best use of

the reliability and predictability of machines.