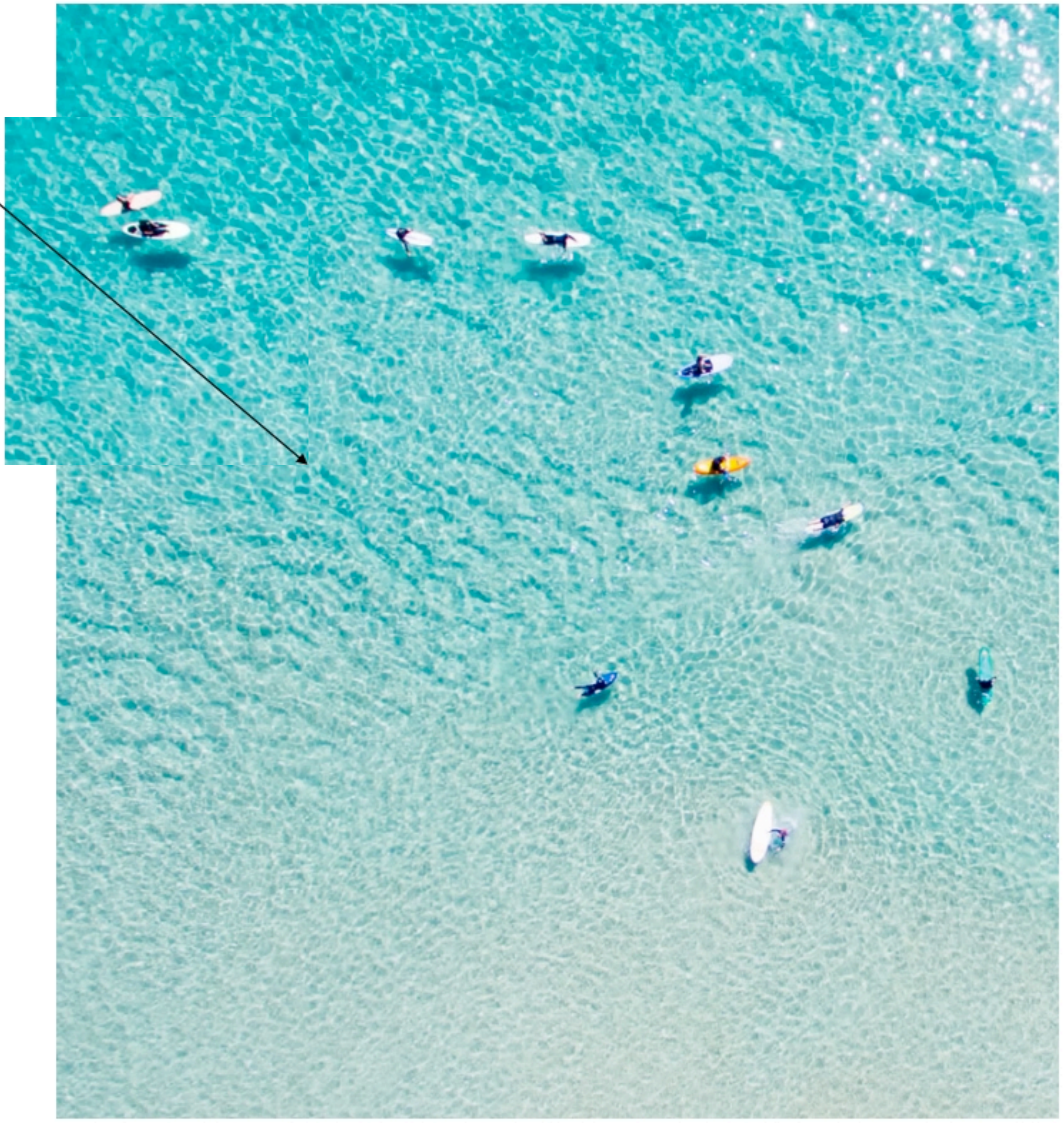
Systems Enterprise Architecture (SEA) : or Systems *Engineering* Architecture



The Next Generation,

based on *Digitization*: quantified engineering, values, digital alignment, AI, Net 3.0, Digital Ontologies, Trinities, ValPlan, Value Agile.

By Tom Gilb

Started 30Aug2020, work in progress, Draft done 12Sep2020
https://LeanPub.com/SysEntArchBook

Systems Enterprise Architecture (SEA) : The Next Generation, based on *Digitization*: quantified engineering, values, digital alignment, AI, Net 3.0, Digital Ontologies, Trinities, ValPlan.

Book Outline:

0.0 Introduction Why am I writing this book at all.

I have long been unhappy with the Enterprise Architecture

methods used by IT people internationally [VR]. I have

taught my ‘Architecture Engineering’ classes for several years

Selling the certifications, for such poorly crafted architecture methods, is a shame, shared by the agile certification game. These methods are an insult to common sense that says we must take qualities and costs far more seriously in real and large systems. Why are so many people fooled?

The core problem as I see it is that the methods taught and used are no longer good enough for the large scale, complex, changing technology. The central idea that is missing is quantification [Q] (of all values, qualities and costs).

Without this tool, too many bad decisions, and costly ones will be made. Quantification is the distinction between arts and crafts on the one hand, and engineering, science, good management, logic, traceability, responsibility on the other hand.

The problem is not new. There comes a threshold of complexity in all disciplines where quantification becomes a necessary tool. The time has come for IT Enterprise Architecture to really make use of quantification, and not just talk about doing it.

I have written many books about this subject, and they are in the references. But they have not influenced EA so it is time to write a book aimed directly at EA culture, so they have no excuses, that this is some other discipline.

This is probably a hopeless case. The only time things will change is when management will only hire and employ people who can serve their organizations interests better, by being more like engineers. Anyway this is my contribution.

Let me be clear that my objective is to change poor practices. So I would be pleased if anybody would like to copy the ideas in this book in whole or part. The ideas are free. But if any reader wants to commercialize the ideas, in methods packages, training, consulting, apps, so that they are spread effectively. That is welcome. I provide a recipe, you can bake the cakes. Nice if you credit sources, I do, but I won’t sue you for forgetting.

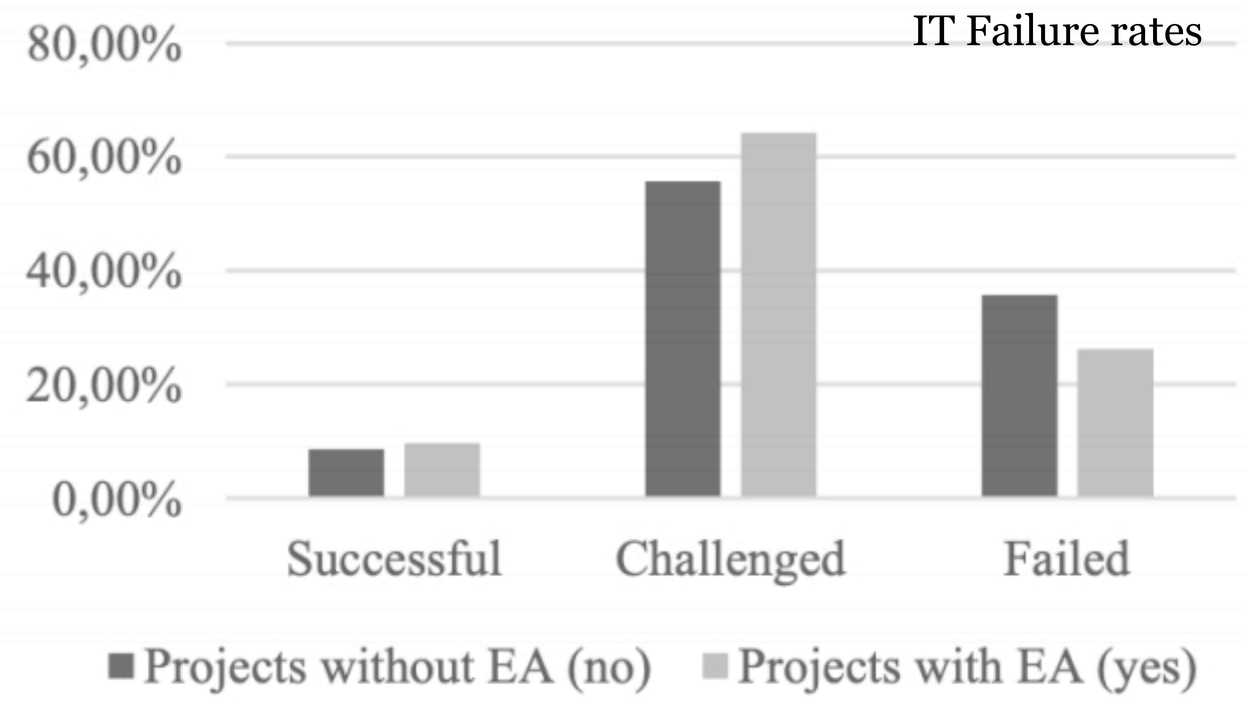
0.0 Introduction Why am I writing this book at all.

Research by The Standish Group already indicates that only 5% of the Grand projects are successful [R2]

Does Enterprise Architecture have significant influence on IT Project Success?

Success rate of large and grand projects with modern resolution

A total of 716 projects qualify as large and grand projects. Of these, 456 projects w~~h~~ere executed without EA and 260 projects with EA.



We saw an increase of successful projects with 11,6%, an increase of challenged projects with 15,3% and a decrease of failed projects with 26,6%.

*Figure 0.0 A [R2] 'IT' is a huge (**95%** for Grand projects) **failure**.*
'EA' does not make results better.

EA is not very influential

"Is enterprise architecture then, a panacea for IT projects? Using the CHAOS database allows us to put the value of EA in perspective of other 'variables', like the project size, agile vs waterfall process, effectiveness and maturity of the project sponsor, among others.

While not conclusive yet, our research indicates it to be of 'moderate' influence, less than having a good sponsor, a small project size, or an agile process, but more than project management frameworks, like PRINCE2.

This is not that unexpected, in our view, as EA predominantly has influence on design, not on the execution of the project."
[R2]

Figure 0.0 B [R2]

*My remark. If the 'design' were clearly related to **quantified values** (which it is NOT in traditional EA), then the design process is a sharper process, and would be expected to deliver values better (my corporate case study experience).*

We also need some continuation interface from EA, into the projects, which 'quantified value requirements' provide.

In addition if the 'agile' processes were equally 'numeric' about value delivery, that will also influence the results. [VA]

0.1 Technical Introduction: The advanced technical components.

Here are the technical ideas, that make SEA different, and better.

10.1. **The architecture planning language, 'Planguage' [CE]** is the systematic framework for SEA. I can also call it SEAL, SEA Language. Make no mistake we are moving to an 'engineering' paradigm. This includes standard practices, with years of practical application such as:

10.1.1 A **Glossary** of about 700 Concepts related to architecture [P4].

10.1.2 About 100 **Principles** of Architecture [CE}.

10.1.3 A large number of specification **Rules**, for good practice in specification of architecture. [P7, CE]

10.1.4 A number of well-defined architecture **processes [CE]**

10.1.5 A defined set of graphical **icons** to express architecture ideas. [P12]

10.1.5 **Systems**: We can manage *anything* here, *not* just IT, but all other related system components such as databases, people, contracts, laws, policies, interfaces, motivation, organization, and non-IT operational systems.

2. **The app 'ValPlan'** (Value Planning): this app (which we have designed and built, and used on our architecture courses [R.ValPlan]. I will use this tool throughout the book. I believe that the digitization of these architecture ideas is one essential step. The basic methods can be practiced without any app, or with spreadsheets. You are welcome to make your own apps. But we will assume the use of this app. I believe that such automation of the architecture process is necessary.

3. SEA is **designed**, and the ValPlan app **using 'itself'** so there is an architectural model (actually several) of it all.

4. SEA is *complete* in *great detail*. But it is also *extendible* by *any user* for any purpose.

5. **Trinity Tools**: https://graphmetrix.com, R.GraphMetrix. There is a set of very-advanced data management and AI tools, being built and released as I write (30 Aug 2020). We are heavily involved in investment and management. It has these [CE] methods as inspiration and framework. When we have more increments released, we can add to the SEA method in significant ways. In short we will be able to integrate the enterprise IT architecture, with all other detailed real-time information, about the organization. 'Property' and 'Logistics' are the first products, along with existing property construction apps.

There are several exciting technical capabilities, already working, for intelligent alignment of all corporate data. The capability of intelligently using very-old data sources, and almost any interfaces, and programs, is itself interesting for IT architecture.

This is so near-future availability, and so exciting, that it is the next generation IT Architecture, so we are going to include it as an assumption, and will detail it, as it rolls out to the public.

Some key words, Artificial Intelligence, Automated Ontologies (Trinity Relations), Owning your own data, but sharing it safely, Object-oriented data (data finds relationships automatically). Rapid real-time adaptation to major and minor changes, to data and program components.

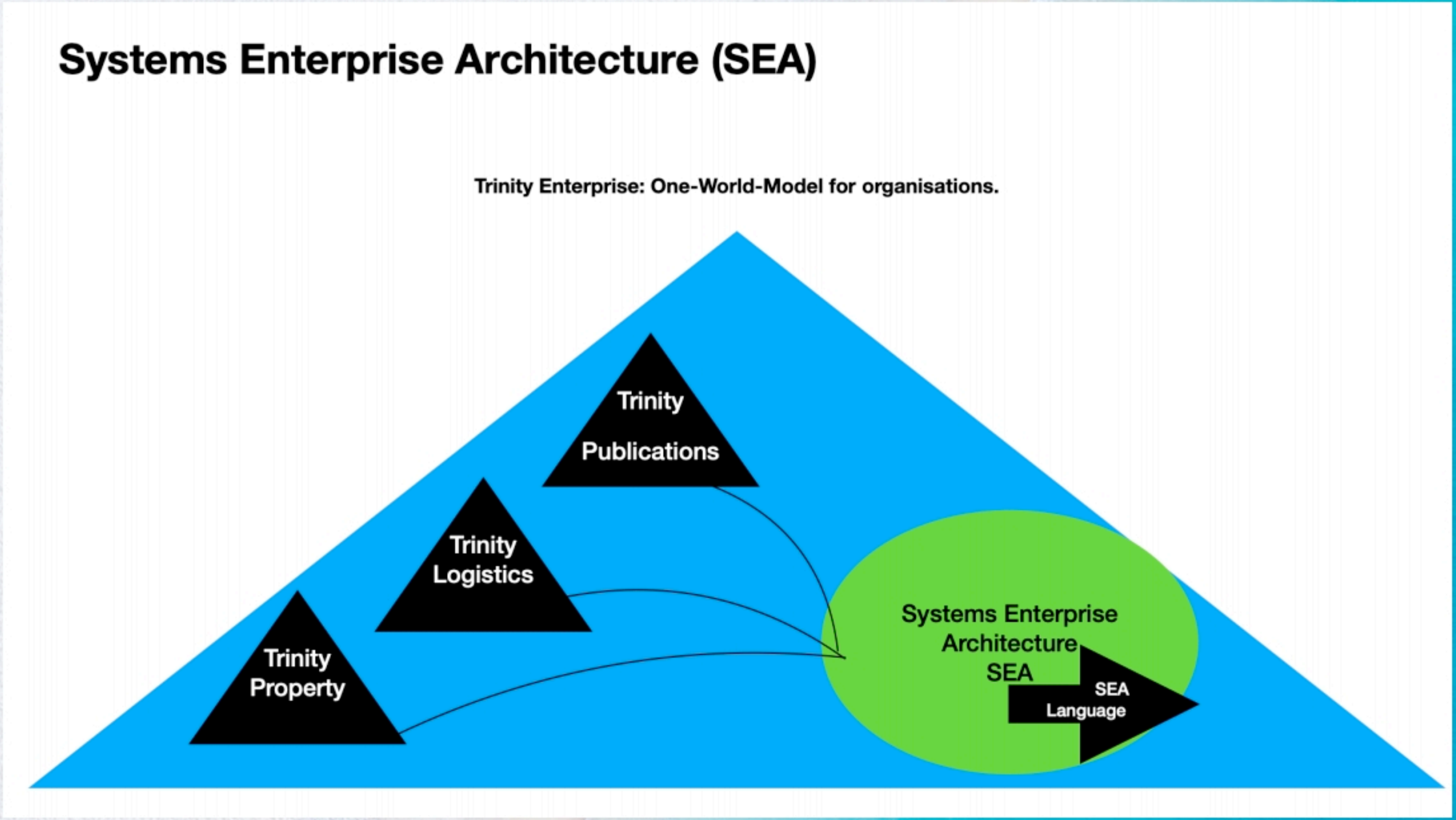

Systems Enterprise Architecture (SEA)
Trinity Enterprise: One-World-Model for organisations.
Trinity Publications
Trinity Logistics
Trinity Property
Systems Enterprise Architecture SEA
SEA Language

Figure 0.1 The Trinity Enterprise vision. Enterprise Architecture Engineering of Systems, aligned with all other organizational data.[R.GraphMetrix].

1.0 Value-Driven Architecture Here are our core SEA principles:

1. ***Stakeholders* determine values, and requirements**

2. **Value requirements determine *necessary* architecture**

3. **Costs and constraints determine *possible* architecture, and priority**

These subjects are the first part, the first 3 Chapters of this book. They lay the basis for the next part of the SEA architecture process: how we evaluate, decompose, and deliver the architecture.

Every detail is connected to every other detail, somehow. But for purposes of learning, we are dividing and sequencing the explanation of the ideas. Forgive me if some ideas pop up a little bit earlier than formally planned.

Stakeholders set the conditions for architecture.
Architecture must satisfy stakeholder needs and resources.

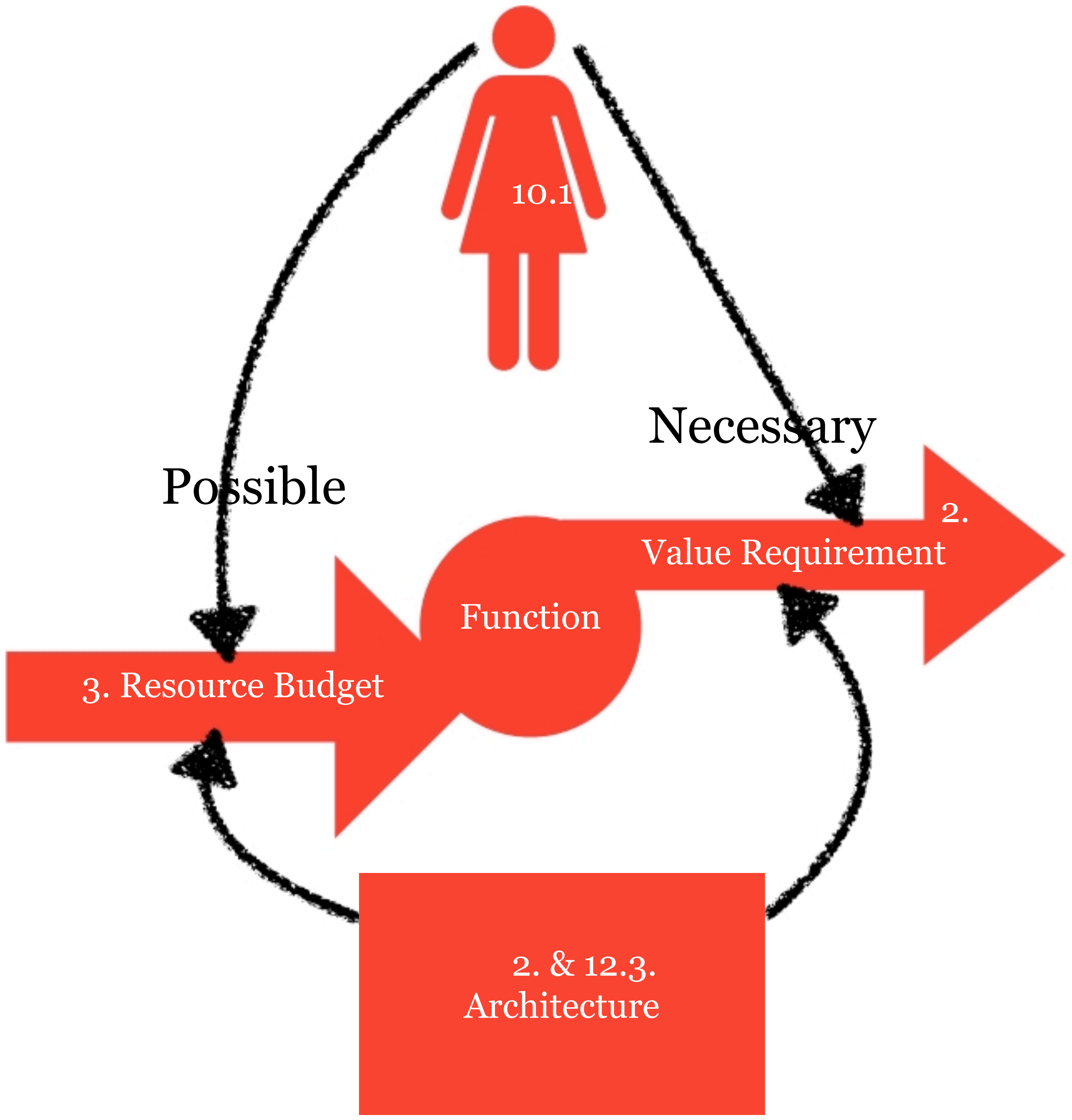


Figure 1.0 The interface between Stakeholders and Architecture are their values and resource constraints.

1.1 Stakeholders

Stakeholders are far more than users and customers.
Some are inhuman, some are downtrodden.
But they influence our architecture.

Stakeholders are the entities that determine our architecture requirements for us.

They determine the content, the level, and timing of a wide array of their non-financial values, and of course some of their financial values.

Stakeholders determine the resources of many kinds, in the short and longer term which we can access, to deliver the values they desire,

Stakeholders are the source of constraints, or limits on our architecture.

In the large systems we need to manage there are many stakeholders, let us say 50 to 500. And these are just general types, like nurse, or contract. In addition to which each specific instance of real stakeholders has different requirements from the general stakeholder instance.

The task of mapping stakeholders, their needs, not to mention future, yet unknown, stakeholders and needs, is impossibly large if we seek perfection. So we need to find a balance, in stakeholder analysis, to suit the architecture task, and a balance which pays off.

Stakeholders-and-their-needs is not a one time, up-front architecture analysis concern. It is *continuous* for the lifetime of the architecture. That is the reason stakeholders need to be specified *digitally*, and connected to their values, and then to the architecture. All digitally. *Not* by 'static paper diagrams'.

Existing stakeholders, and their needs, will also *continuously* change. We need to keep updated, ahead of the game, digitally, and to sense the consequences of change, digitally, so we can *adjust our architecture* correspondingly.

As you see I am staring to make the argument for the digital tools, ValPlan and GraphMetrix. Non-automated methods will not work accurately, cost-effectively and quickly enough.

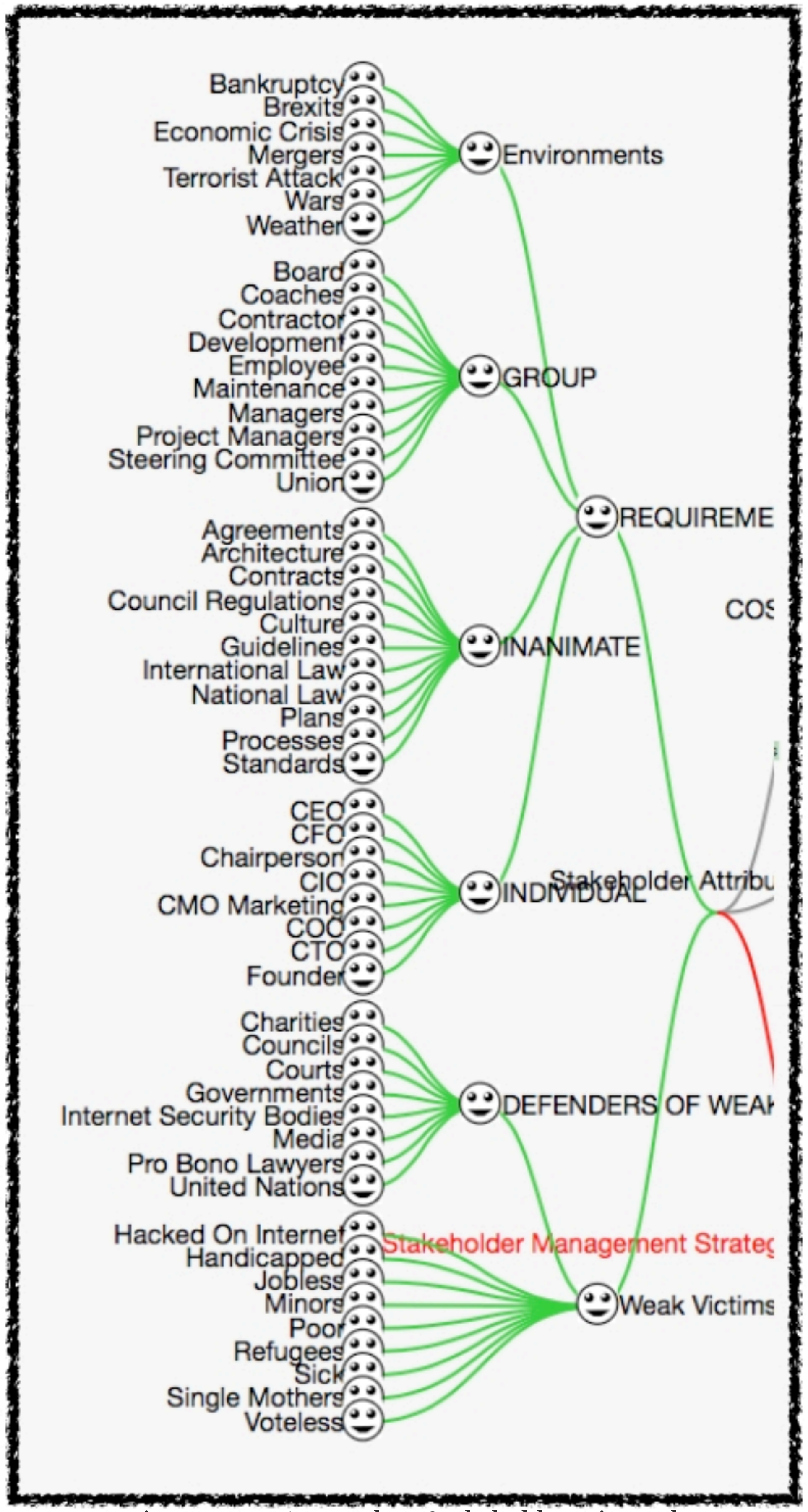


Figure 1.1 B. A Template Stakeholder Hierarchy

A stakeholder can have many value requirements, a value requirement can have many stakeholders. A stakeholder without a value requirement is not a real stakeholder, or we are missing a requirement for them. All Values must have some supporting architecture. Charts like this are possible as a consequence of Planguage digitization (ValPlan app). Things are a little messy at this early stage (2-3 days work) of drafting architecture.

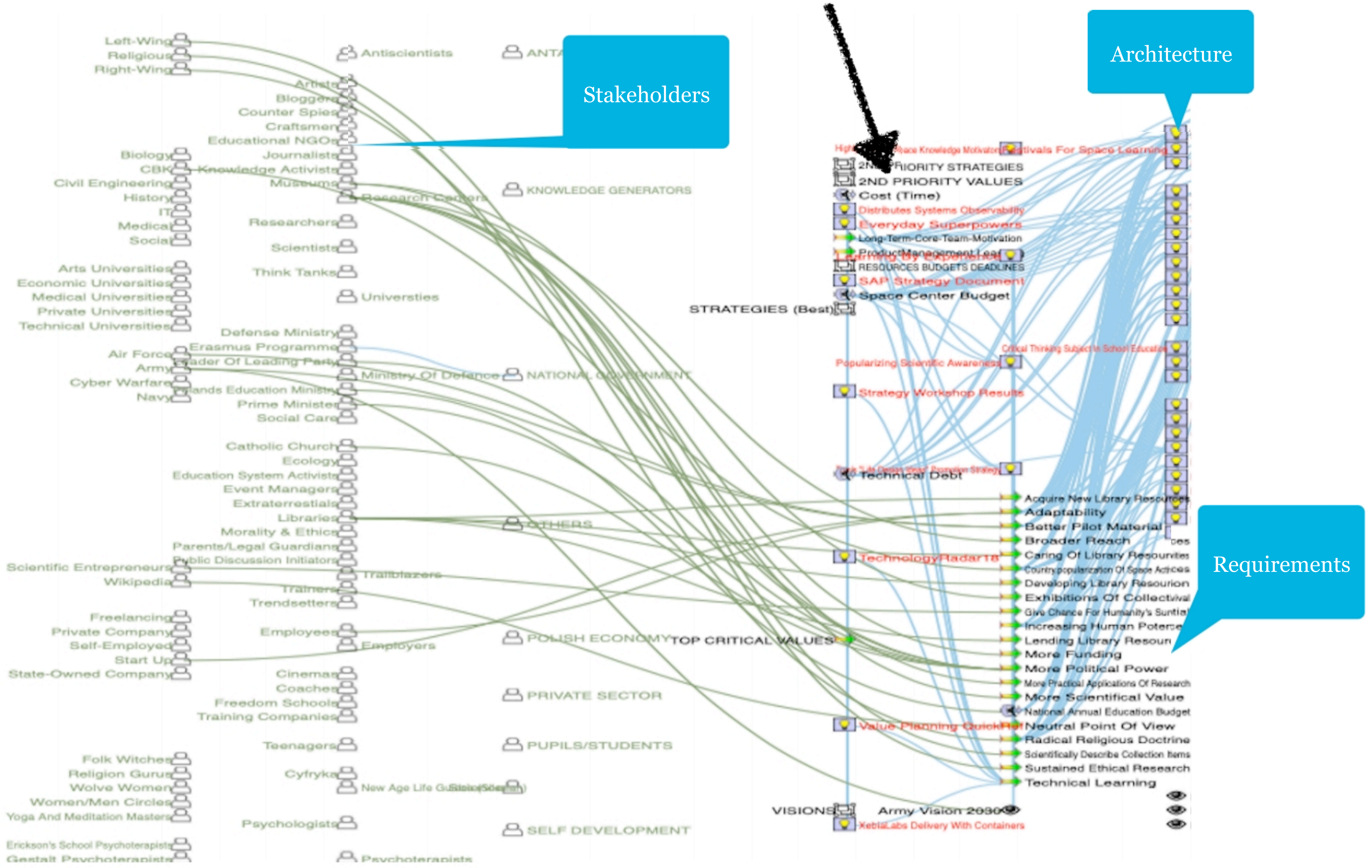


Figure 1.1 B ValPlan diagram, mapping from digital object relationship data, the relations between stakeholders, values, and architecture (Knowledge Project Poland, Masterclass 2018). Planguage and ValPlan are strong on continuous real-time tracking, of critical relationships.

1.2 Values (of stakeholders)

'*Values*', as a concept is easily misunderstood. But it is a very useful concept. So, let us deal with it clearly.

The major derailment is that people think of *money* value, when they hear the term *value*. But, we are using it in a much *wider* sense. We mean *everything, that stakeholders value.*

We mean a very broad range of human values, where money is in the picture, but is not the main idea. Perhaps a simple start is the United Nations 17 Sustainable Goals [SP], like about poverty, hunger, education. Very important stuff, and it costs money. But money is not the main point.

There is no simple fixed quantity of possible value statements (Value Objectives, Value Requirements) from stakeholders. Value specifications are, as we shall see (see section 10) , *infinite* in their detail. They are *whatever* the stakeholder needs, and the stakeholder circumstances, are both changing, and infinitely complex.

We do not give up, at this infinite *value definition* task. It is the *basis* for our architecture. We do the best we can, *early*.

We need to do a pretty good job at capturing and prioritizing stakeholder requirements. Then we modify requirements, as needed, in the future. But this ongoing complexity, demands *smart digital systems*, which can *track* changes, keep the complete *overall* picture, and *all relations* intact, and help us see the *need* for architectural *change*, in good *time*.

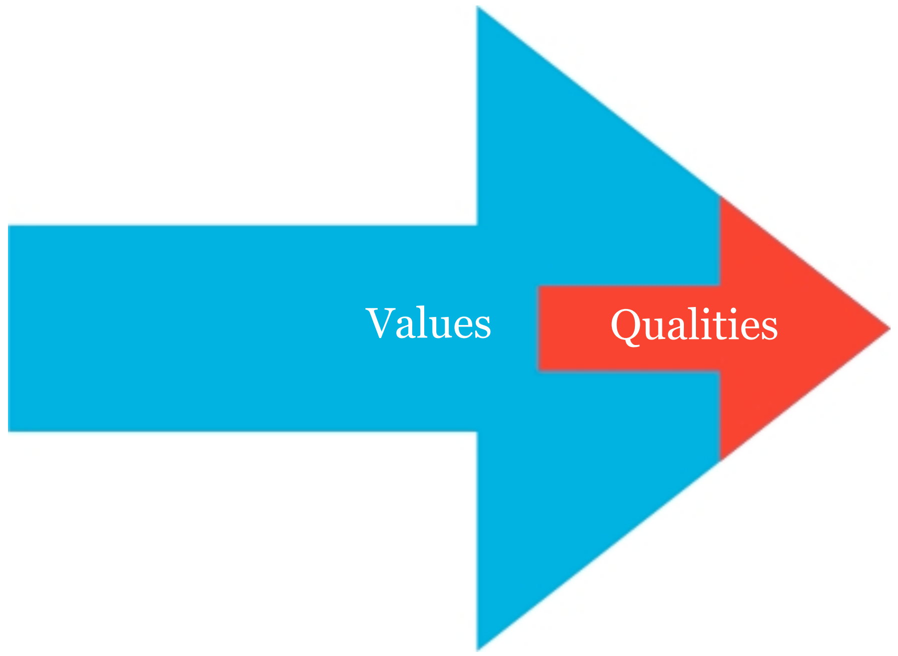


Figure 1.2 A:
Sometimes the value (how good)
is a particular system quality
(like Security, Usability). i . e . How well.

Some other values are not qualities,
Like 'Performance', 'throughput' (i. e. how much)

The architect task is to 'find a means to deliver the values'

For detail on how to specify quantified Values see this book Part 10.1

1.2 Values (of stakeholders): Defined as Planguage Concept. ‘*269’

Value **Planguage Glossary Concept: *269**

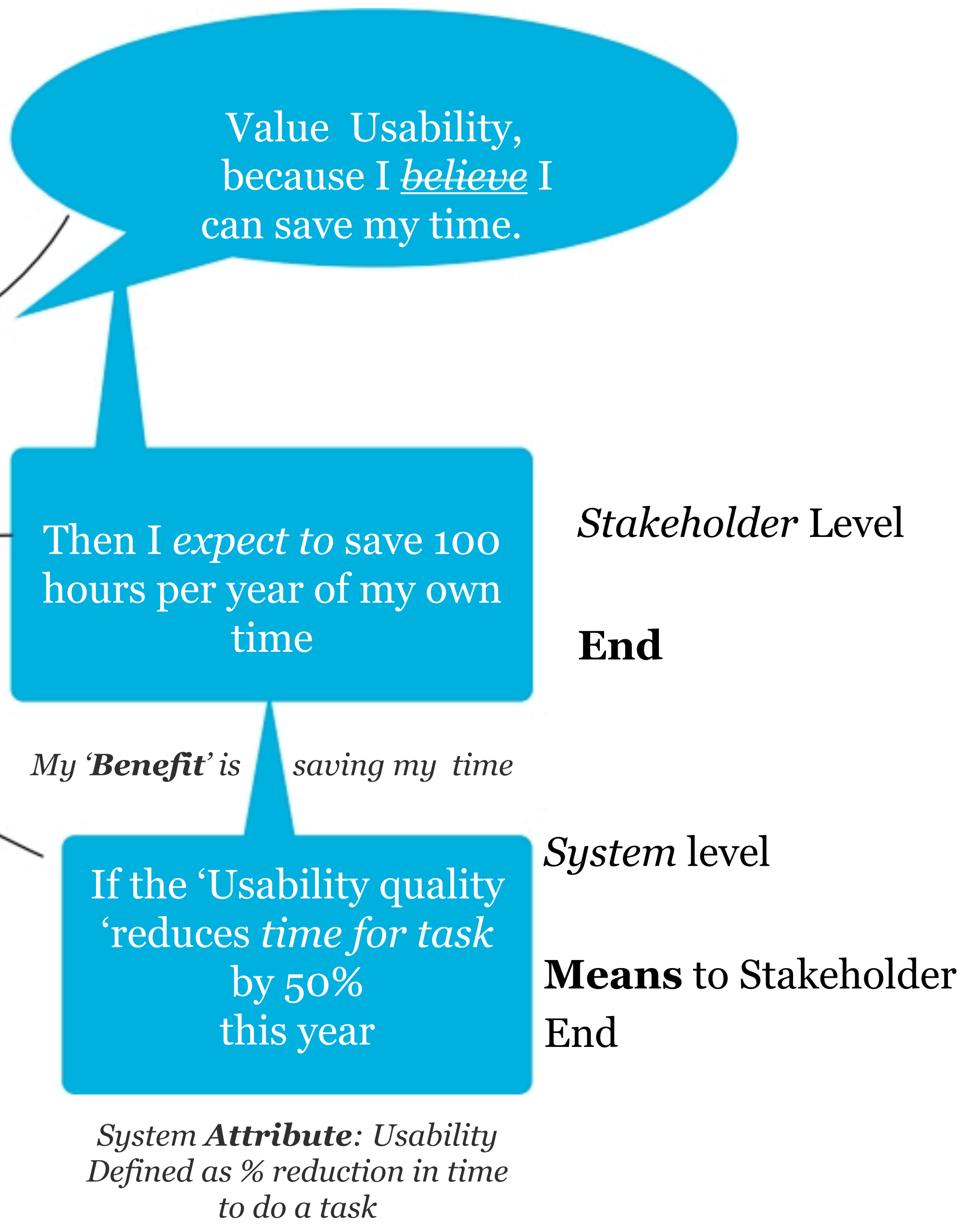
Value is *perceived* benefit:

that is, the benefit we think we will get from something.

Glossary Concept Notes:

1. Value is the potential consequence of system attributes, for one or more stakeholders.
2. Value is not linearly related to a system improvement: for example, a small change in an attribute level could add immense perceived value for one group of stakeholders for relatively low cost.
3. Value is the perceived usefulness, worth, utility or importance of a defined system component or system state, for defined stakeholders, under specified conditions.

   “One man’s meat is another man’s poison.” Old proverb
4. ‘Benefit’ is when some perceived value is actually produced by, a defined system.
5. Value is relative to a stakeholder: it is not absolute. Quality, for example, is stated in terms of the objective level of ‘how well’ a system performs, irrespective of how this level is appreciated by any stakeholders. Some defined levels of quality only have a value to some stakeholders. The same is true for all attributes. There are many Planguage ways of indicating that a stakeholder values an attribute. These include using Value, Stakeholder, Authority, Impacts, and Source parameters.


Value Usability, because I *believe* I can save my time.
Then I *expect to* save 100 hours per year of my own time
Stakeholder Level
End
*My ‘**Benefit**’ is saving my time*
If the ‘Usability quality ‘reduces *time for task* by 50% this year
System level
Means to Stakeholder End
*System **Attribute**: Usability Defined as % reduction in time to do a task*

“Nowadays, people know the cost of everything and the value of nothing.”

Oscar Wilde.

For detail on how to *specify* quantified Values see this book Part 10.1

Synonyms: Worth *269.
Related Concepts: Benefit *009; Impacts *334; Values *592.

1.3 Costs and Resources

Stakeholder-held Values, translated into specified 'value requirements', define for the architect, the *expected* level that a proposed architecture must deliver. For all architecture options, which meet this level of performance, there are still *some* considerations *before* we can accept, choose, or prioritize an architecture option.

1. Can we *afford* it?

2. Are the *side-effects* ok?

3. Does it violate any specific *constraints*?

The 'can we afford it' *architecture qualification*, is what we are going to discuss now.

In simplistic terms, if the budget is 1 million, and the architecture cost is 5 million. You cannot afford it.

We cannot accept an architecture component, or a complete architecture, just because 'it will deliver a value-level on-time'.

A second consideration, is that there probably is *more than one* resource-constraint (budget limitation). There can be resource budgets for people, money, and time (cap ex). There can be budgets for operational or recurring costs (op ex).

I have been surprised to see that older architecture methods, allow the architect to specify architecture, with no mention or consideration of its numeric real- impact on these budgeted, limited, resources.

This 'resources' omission is no way to manage large-scale architecture. It is only OK when the system is small, and the costs obvious, and are tolerable, 'no matter what'. Otherwise, lack of *resource impact estimation* of architecture ideas, is intolerable, and a sure way, to get 'hit' by resource problems, and even by total system failure.

One important job of the architect, is to keep control of *resource consequences* of architecture decisions.

This is known as '**design to cost**' by engineers ('architect-to-cost' ?). And I am going to suggest an 'agile' version of this which I call *Dynamic* Design-to-Cost, and it is built into my Planguage method [S5, S6]

Serious cost-of-limited-resources estimations, is absolutely mandatory for serious enterprise architecture work. What are *you* practicing in this cost area, today?

If your architecture staff cannot, or will not, do this costing, they are incompetent and dangerous. As a CTO for example, you need to take 'responsibility for costs of architecture proposals'.

1.3 Costs and Resources: of architecture options.

This is a set of cost budgets for the 'resources needed', to deliver the Knowledge Values set above.
Sorry if this level of detail disturbs some people. We are trying to model only the critical-few values, and the critical-few resources. Critical = 'can kill your project'.

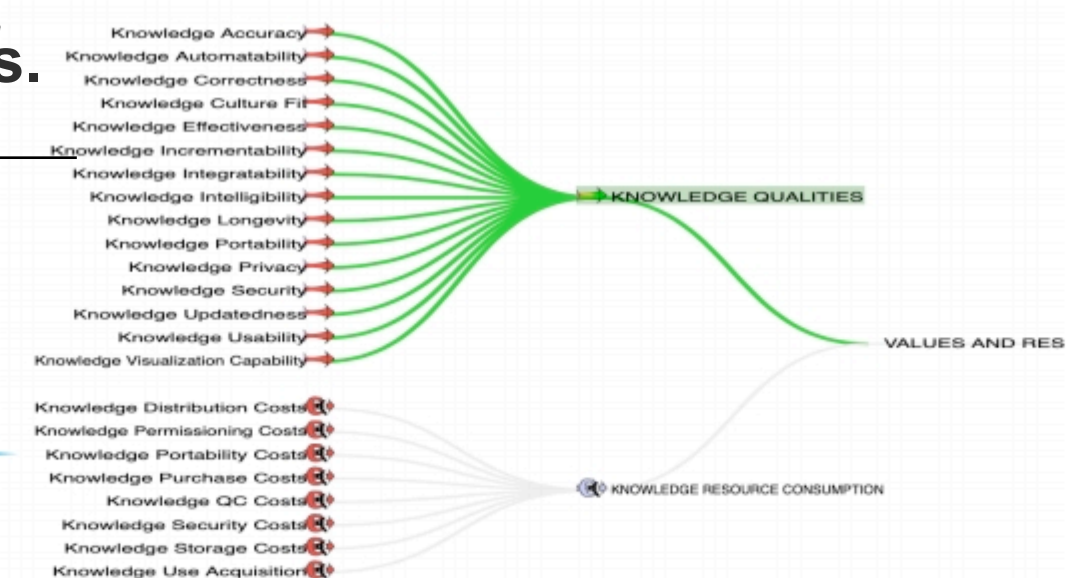
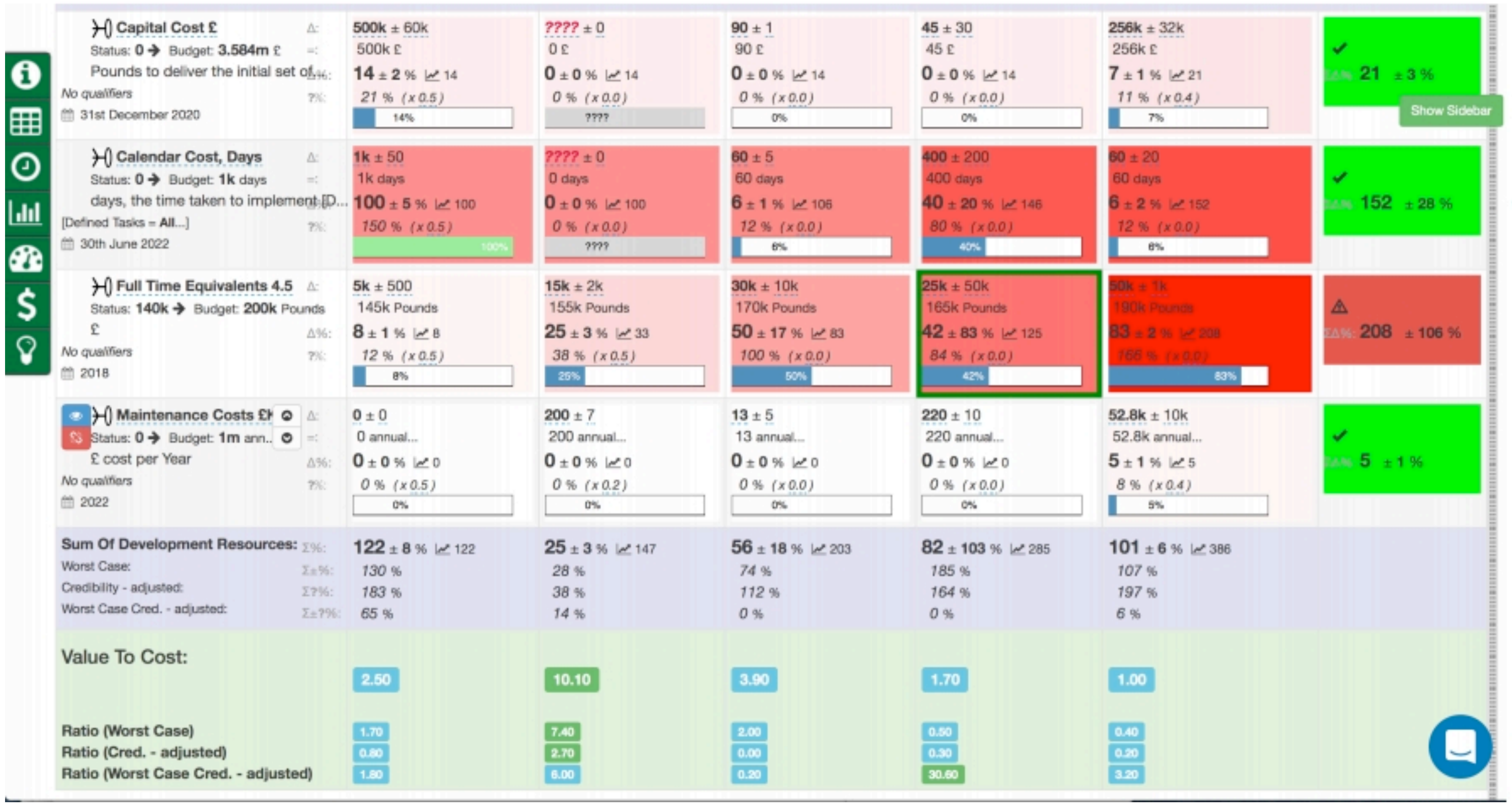


Figure 1.3 Another example of a set of costs for Knowledge systems (ValPlan app, [KEN]).

Evaluating Multiple Costs of a set of architecture options .

Arch 1 ———––Arch 2 ——–- Arch 3 -—-— Arch 4———–– Arch 5



		Arch 1	Arch 2	Arch 3	Arch 4	Arch 5	
Capital Cost £ Status: 0 → Budget: 3.584m £ Pounds to deliver the initial set of... No qualifiers 31st December 2020	Δ: =: Δ%: ?%:	500k ± 60k 500k £ 14 ± 2 % 14 21 % (x 0.5) 14%	???? ± 0 0 £ 0 ± 0 % 14 0 % (x 0.0) ????	90 ± 1 90 £ 0 ± 0 % 14 0 % (x 0.0) 0%	45 ± 30 45 £ 0 ± 0 % 14 0 % (x 0.0) 0%	256k ± 32k 256k £ 7 ± 1 % 21 11 % (x 0.4) 7%	✓ ΣΔ% 21 ± 3 % Show Sidebar
Calendar Cost, Days Status: 0 → Budget: 1k days days, the time taken to implement [D... [Defined Tasks = All...] 30th June 2022	Δ: =: Δ%: ?%:	1k ± 50 1k days 100 ± 5 % 100 150 % (x 0.5) 100%	???? ± 0 0 days 0 ± 0 % 100 0 % (x 0.0) ????	60 ± 5 60 days 6 ± 1 % 106 12 % (x 0.0) 6%	400 ± 200 400 days 40 ± 20 % 146 80 % (x 0.0) 40%	60 ± 20 60 days 6 ± 2 % 152 12 % (x 0.0) 6%	✓ ΣΔ% 152 ± 28 %
Full Time Equivalents 4.5 Status: 140k → Budget: 200k Pounds £ No qualifiers 2018	Δ: Δ%: ?%:	5k ± 500 145k Pounds 8 ± 1 % 8 12 % (x 0.5) 8%	15k ± 2k 155k Pounds 25 ± 3 % 33 38 % (x 0.5) 25%	30k ± 10k 170k Pounds 50 ± 17 % 83 100 % (x 0.0) 50%	25k ± 50k 165k Pounds 42 ± 83 % 125 84 % (x 0.0) 42%	50k ± 1k 190k Pounds 83 ± 2 % 208 166 % (x 0.0) 83%	⚠ ΣΔ%: 208 ± 106 %
Maintenance Costs £k Status: 0 → Budget: 1m ann.. £ cost per Year No qualifiers 2022	Δ: =: Δ%: ?%:	0 ± 0 0 annual... 0 ± 0 % 0 0 % (x 0.5) 0%	200 ± 7 200 annual... 0 ± 0 % 0 0 % (x 0.2) 0%	13 ± 5 13 annual... 0 ± 0 % 0 0 % (x 0.0) 0%	220 ± 10 220 annual... 0 ± 0 % 0 0 % (x 0.0) 0%	52.8k ± 10k 52.8k annual... 5 ± 1 % 5 8 % (x 0.4) 5%	✓ ΣΔ% 5 ± 1 %
Sum Of Development Resources: Worst Case: Credibility - adjusted: Worst Case Cred. - adjusted:	Σ%: Σ±%: Σ?%: Σ±?%:	122 ± 8 % 122 130 % 183 % 65 %	25 ± 3 % 147 28 % 38 % 14 %	56 ± 18 % 203 74 % 112 % 0 %	82 ± 103 % 285 185 % 164 % 0 %	101 ± 6 % 386 107 % 197 % 6 %	
Value To Cost:		2.50	10.10	3.90	1.70	1.00	
Ratio (Worst Case)		1.70	7.40	2.00	0.50	0.40	
Ratio (Cred. - adjusted)		0.80	2.70	0.00	0.30	0.20	
Ratio (Worst Case Cred. - adjusted)		1.80	6.00	0.20	30.60	3.20	

*Figure 1.3 B. Four different resource budgets (left) and estimates of costs for 5 architecture options (middle 5 columns) This is the **resources section**, of an Impact Estimation Table, in the ValPlan app. More later, but the point is we can digitize keeping track of 4 resources, and 5 different architecture options.*

This level-of-detail might be overwhelming, for people who never give costs a thought. See section 12 here for details.
This is an *engineering* approach, so lots of numbers, and some of them are about the credibility and uncertainty of the cost estimates.

For now, focus on the 20 rectangular bars, with % of Budget, as estimated cost. Start with the upper left bar, 14%

The ???? Is a 'known unknown' cost for the architecture.

For detail on how to specify quantified Cost and Resources see this book Part 10.1

1.4 Constraints, Binary and variable

There are two main types of planning specification 'constraints'.

Binary Constraints *(see [B1] for more detail on non-quantifiable back- white requirements) which are* are of the do-or-die type. You either do them, or you do not. You can test-or-ascertain, whether they are planned, or implemented, or *not*. They are important for architecture validation, and architecture choice. Architectures should not violate binary constraints, without conscious valid permission.

Examples: 'legal requirements' (must conform to EU Laws), and 'Design Constraints', 'You must use Sustainable Products'. Of course in both cases there may be some leeway, wiggle room. But that depends how you define 'Legal' and 'Sustainable'. There *are* black and white areas, even though the *white* might be a bit grayish or dirty. See detail here in 10.3.

Scalar Constraints: are 'potentially' (always possible actually) *quantifiable*. Meaning 'you can define a useful Scale', and define a 'constraint point' on that Scale. 'Tolerable level is 50%'.

Simple examples are: 'you cannot withdraw more than €400 from the cash machine at once.', or 'no actual usage, or payment is due if ,or whenever, the mobile cellphone network is unavailable less than 50% of an hour'.

Scalar Constraints are *not* the **target** requirements we want to achieve, for 'Success' (see `Goal') . They define the border minimum and/or maximum level, permitted for defined purposes (safety, payment, operation, etc.).

Of course all the tools that apply to all other Planguage 'Performance Levels' apply. Such as: Use of Scales. Use of Tags. Multiple instances. Use of specified Conditions, Dates, Ranges of Numbers, Source Annotation, Justification, Issues notation, Assumptions specification, Connection to specific Stakeholders, and more. We have not discussed all these Planguage tools *yet* in this book (see Section 10.), but they are all in other books, such as Competitive Engineering [B1] and Value Planning [B2].

The **Scalar Constraint**, as with most all other Scalar Points, not only applies to stakeholder Values (performance, qualities, benefits, good stuff). They also apply to a subject we have not treated yet, **Resource Planning**. For example, there may be a Deadline of '1 year from project start'. But there can be a *Constraint* of 2 months *over* the deadline, which triggers automatic Fines or contract cancellation.

Scalar constraints are the *first consideration (priority)* in any strategy planning or architecture. We need to choose and evaluate strategies, so that it is *unlikely* we will violate a Constraint. Later, we will manage the strategy design process, to reach all Goals, using a minimum of resources. But step one is: d*o not violate the constraints*. There is no point in 'optimizing' an illegal, unavailable system.

Obviously, Scalar Constraints are good tools to use in contracting, safety, operations, handover, delivery, and other such critical areas.

I am constantly surprised by how often I see planning methods, that focus all attention on the planned target level (Goal) but have *no* discussion, practice, teaching, or understanding of a Scalar *Constraint* of any type.

That includes methods like for example Balanced Scorecard, Quality Function Deployment [B5], and the like, which at least try to quantify the main Goal 'sometimes'. Not always! Other methods of thought, that do not *even try* to quantify critical values *at all*, we need not discuss here. Total Failure of such methods. [B5, B3, B12].

1.4 Constraints, Binary and variable

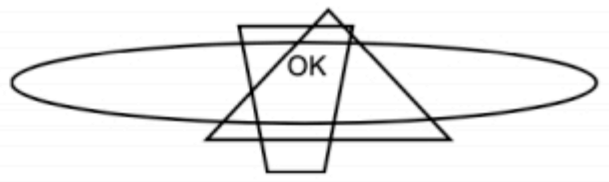


Figure G3
Constraints impose restrictions on both other requirements and designs. But the remaining space ('OK') gives considerable freedom to set more-exact requirements, and to specify more-exact designs.

Figure 1.4 A [Source CE

All constraints, by definition, limit the possible architecture choices

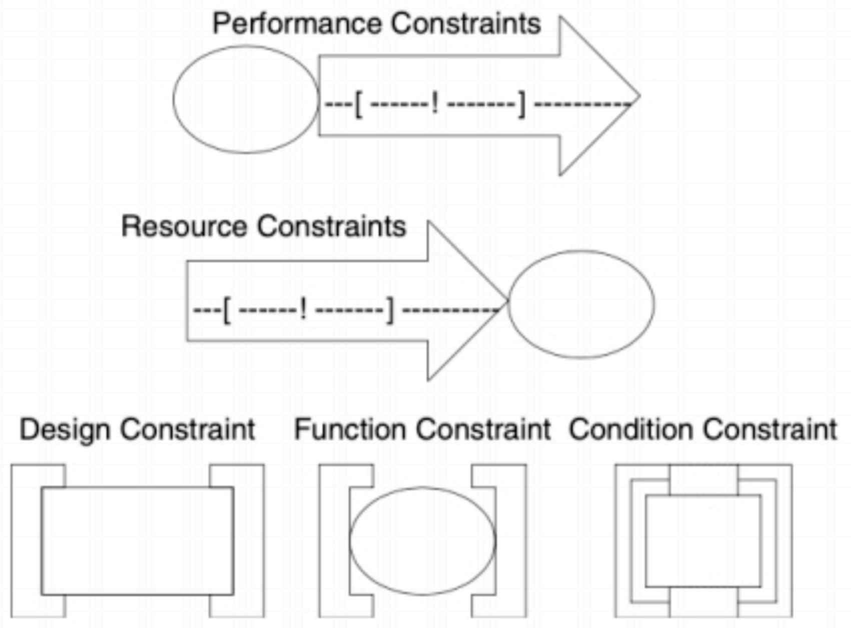


Figure G4
Drawn Icons for Constraints.

Figure 1.4 B [Source CE]

There are many different types of constraints [P4, CE] and as said,
*they **consciously restrict architecture choices**.*

*Keep in mind that **constraints are set by stakeholders**,*
*with **limited priority and power**.*

So, there is always the possibility that
*a **constraint can be modified, or ignored***
if some higher priority requirements conflict with it.

*So constraints are **not static or absolute.***

But once specified, they need to be systematically dealt with.

If constraints are lower priority, changed, or deleted,
the reasons need to be specified in writing
on the written specification,
so people understand,
see the reasoning,
can argue against decisions,
and can review the current decision.

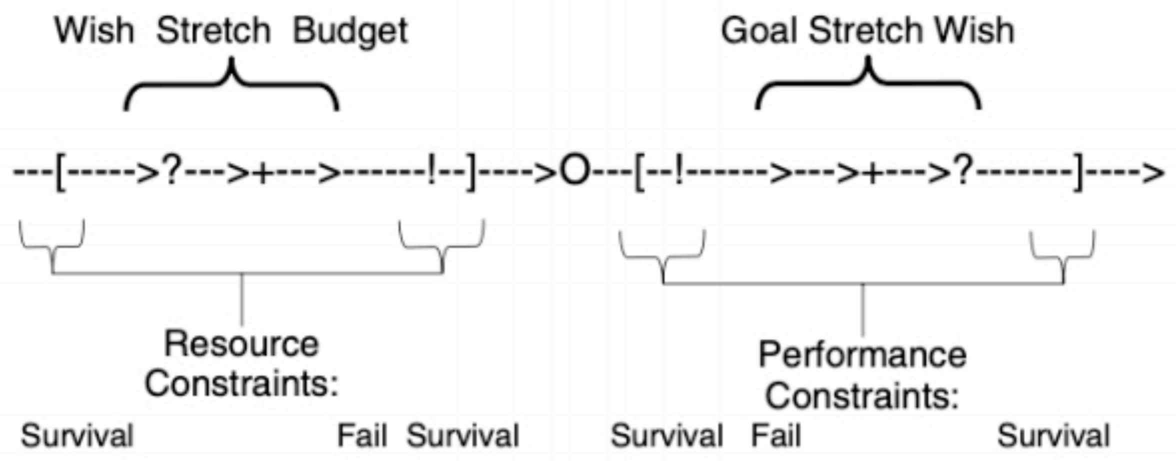


Figure G31
Shows scalar targets can be specified for both performance and requirements.

Figure 1.4 C. [Source CE]
*Scalar constraints (too hot and too cold, or too little finding, unprofitable funding, for example, serve as one of several tools, to help us **determine priorities,** for architecture choices.*

1.5 Side Effects, a constraint, and opportunity

When an architect is focussing on 'one quantified value', or quality, let us say 'Security', and trying to think about possible security architecture ideas; they can pick some ideas which *seem* to satisfy the Security requirements. Good start!

But, then we have to ask 2 basic questions:

1. Can we *afford* the Security architecture?

2. What are the *'side-effects'* on *all other* 'critical stakeholder requirements'. For example, does the architecture idea threaten 'Usability', or 'Performance Speed'?

I have noticed that conventional EA methods have no real systematic approach to this. They hardly bother with any clear requirements, for a start. I could not find anything about EA and side-effects doing internet searches. Yet we all know how important side-effects are in medicines! Side effects can be very unpleasant and even kill you.

There are two different processes the architect can deal with for handling side effects.

10.1. **ESTIMATION**: During the design phase, by estimating side effects on quantified values and costs [Figure 1.5 B]. The estimations are rough, order of magnitude, and preferably based on some sort of experience, somewhere. See Part 12 for detail.

A practical side-effect measurement cycle.

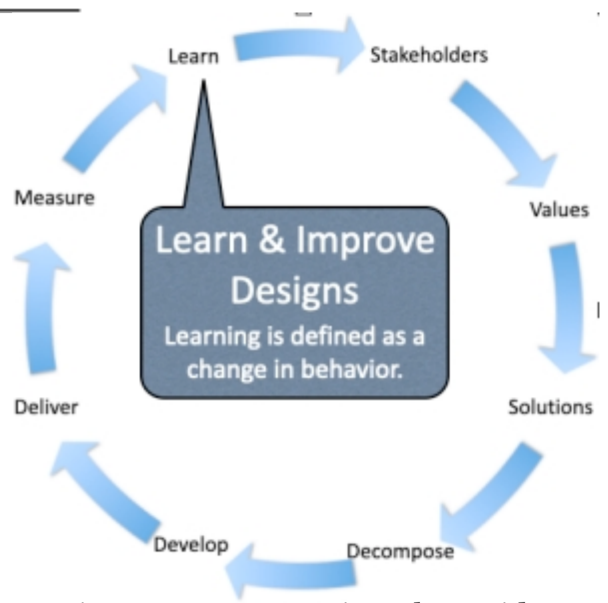


Figure 1.5 A. Learning about side-effects, as a result of an Evo [CE] agile cycle, with measurement feedback, on potential side-effect values and costs.

It is the job of an architect to follow up incremental measures, and take architecture action; when unforeseen side effects occur. [P3.3]

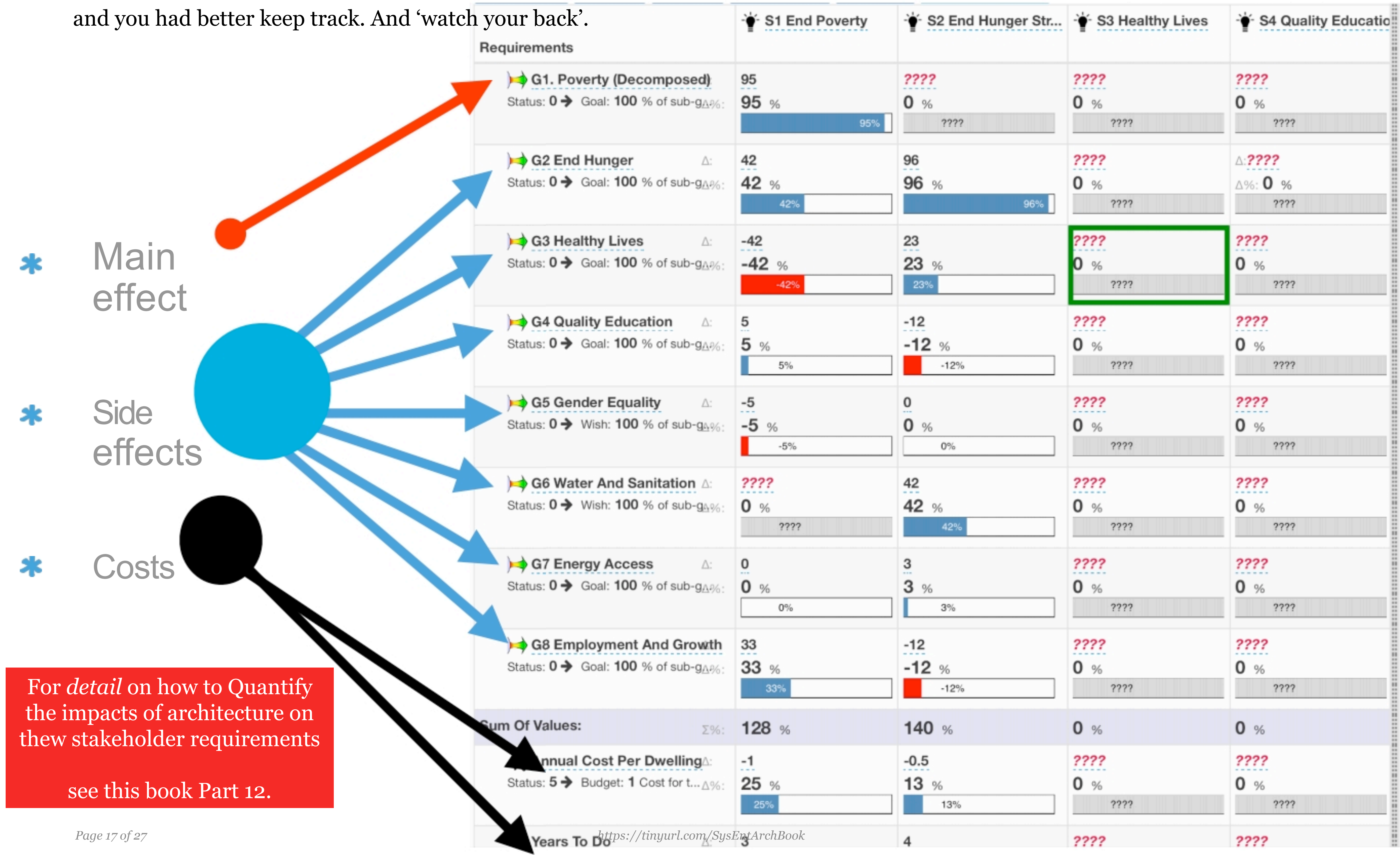
Same principle applies to building architects!

This is theoretical, but better than *ignoring* side effect possibilities, and then making bad architecture *choices*, with perhaps costly *changes*; too late.

2. **MEASUREMENT**: when a small, trial-size, chunk of an architecture idea, is delivered and integrated into a real system, maybe at a small scale, then it is possible to measure, and observe side-effects and unusual costs. Successful project methods (100% on time, under budget), like Cleanroom, charge the architect with *fixing deviations* from needs-and-expectations. [p3.3].

1.5 Side Effects, a constraint and opportunity

Figure 1.5 B. ‘Value Side-effect’s and costs. This is your *first peek* at a **major architectural tool**, an **Impact Estimation Table** (**IET**). In this case, 4 architectures (strategies) are rated (estimated), for *potential impac*t on the ‘9 UN Sustainability Goals’. More later about this method (Part 12). But, I pulled it out, to show the idea of ‘side effects', and ‘costs’. Your architecture impacts all these, and you had better keep track. And ‘watch your back’.



Requirements	S1 End Poverty	S2 End Hunger Str...	S3 Healthy Lives	S4 Quality Educatio
G1. Poverty (Decomposed) Status: 0 → Goal: 100 % of sub-g Δ%:	95 95 % 95%	???? 0 % ????	???? 0 % ????	???? 0 % ????
G2 End Hunger Δ: Status: 0 → Goal: 100 % of sub-g Δ%:	42 42 % 42%	96 96 % 96%	???? 0 % ????	Δ:???? Δ%: 0 % ????
G3 Healthy Lives Δ: Status: 0 → Goal: 100 % of sub-g Δ%:	-42 -42 % -42%	23 23 % 23%	???? 0 % ????	???? 0 % ????
G4 Quality Education Δ: Status: 0 → Goal: 100 % of sub-g Δ%:	5 5 % 5%	-12 -12 % -12%	???? 0 % ????	???? 0 % ????
G5 Gender Equality Δ: Status: 0 → Wish: 100 % of sub-g Δ%:	-5 -5 % -5%	0 0 % 0%	???? 0 % ????	???? 0 % ????
G6 Water And Sanitation Δ: Status: 0 → Wish: 100 % of sub-g Δ%:	???? 0 % ????	42 42 % 42%	???? 0 % ????	???? 0 % ????
G7 Energy Access Δ: Status: 0 → Goal: 100 % of sub-g Δ%:	0 0 % 0%	3 3 % 3%	???? 0 % ????	???? 0 % ????
G8 Employment And Growth Status: 0 → Goal: 100 % of sub-g Δ%:	33 33 % 33%	-12 -12 % -12%	???? 0 % ????	???? 0 % ????
um Of Values: Σ%:	128 %	140 %	0 %	0 %
nnual Cost Per Dwelling Δ: Status: 5 → Budget: 1 Cost for t... Δ%:	-1 25 % 25%	-0.5 13 % 13%	???? 0 % ????	???? 0 % ????
Years To Do Δ:	3	4	????	????

For *detail* on how to Quantify the impacts of architecture on thew stakeholder requirements

see this book Part 12.

2. Architecture Efficiency

The Efficiency of an architecture is the value**s** it gives, in relation to the cost**s** it incurs. The Architecture Efficiency can be stated, using a 'given set of values', and a 'given set of costs'.

The Architecture Efficiency is independent of targets, constraints. But AE (Architecture Efficiency): (but it can be *related* to them, by comparison, if we wish)

1. **PROFITABILITY:** AE is a way of thinking about the profitability of the architecture
2. **GOOD:** AE is a way of defining a 'good' architecture.
3. **SKILL:** AE is a reflection of the skill of the architect.
4. **COMPARISON:** AE is a way of *comparing* architectures, and selecting alternatives.
5. **PRIORITY**: AE is a way of *prioritizing* sub-architectures, in terms of early delivery to a project, or system, so as the conserve resources; by selecting low resource options first.
6. **RESOURCE EVALUATION**: AE, *forces* architects to consider and document, the various *resources, money.time, people* and more; in the short-term (up front) and longer-term (annual, maintenance, repair, recovery, backup, recurrent lifetime costs).
7. **VALUE AND COST SCALES AS %**: The *Architecture Efficiency* can be calculated using an Impact Estimation Table [B1], and by also *expressing the Value ideas as a percentage* of distance from some status level, to some constraint or target level. See more detail in the IET details in this book (Part 12).
8. **RISK QUANTIFICATION**: The Architecture Efficiency (ArchEff ?) can and shoud be modified with the *risk factors* for the *estimates*, or the *measures* taken. Built-in to IET (Figure 2.2), are **two Risk factors**: the **± uncertainty**, and the **Credibility** (0.0 to 1.0) of the estimate, or measure. ValPlan automatically makes use of these to modify the Values/Costs ratio to something nearer the truth [Figure 2.2 A and B].
9. **BALANCE TOOL:** This **V/C efficiency ratio** is also a tool to keep a reasonable architecture *balance*. There is no point in maximizing architecture choices, in order to improve *values*, if the *costs increase disproportionately*. And, there is no point architecting to maximize a *single* value, if there are disproportionate side-effects on the *other* critical values (side effects).
10. **SEQUENCE DEPENDENCY:** the incremental effects of inserting a sub-architecture, in a value-delivery step, is dependent on the system state, and on previous increments. The effects can change and be corrupted by *subsequent* increments of sub-architectures. Tricky nature.

Sub-architecture : is any architecture component, which has been decomposed from any larger architecture specification. Sub-architectures are particularly useful in agile architecture, where we need small frequent deliveries, and feedback from architectures. We can also prioritize the most 'efficient' architectures.

2. Architecture Efficiency. ∑V/∑C x Risk. Picture of AE

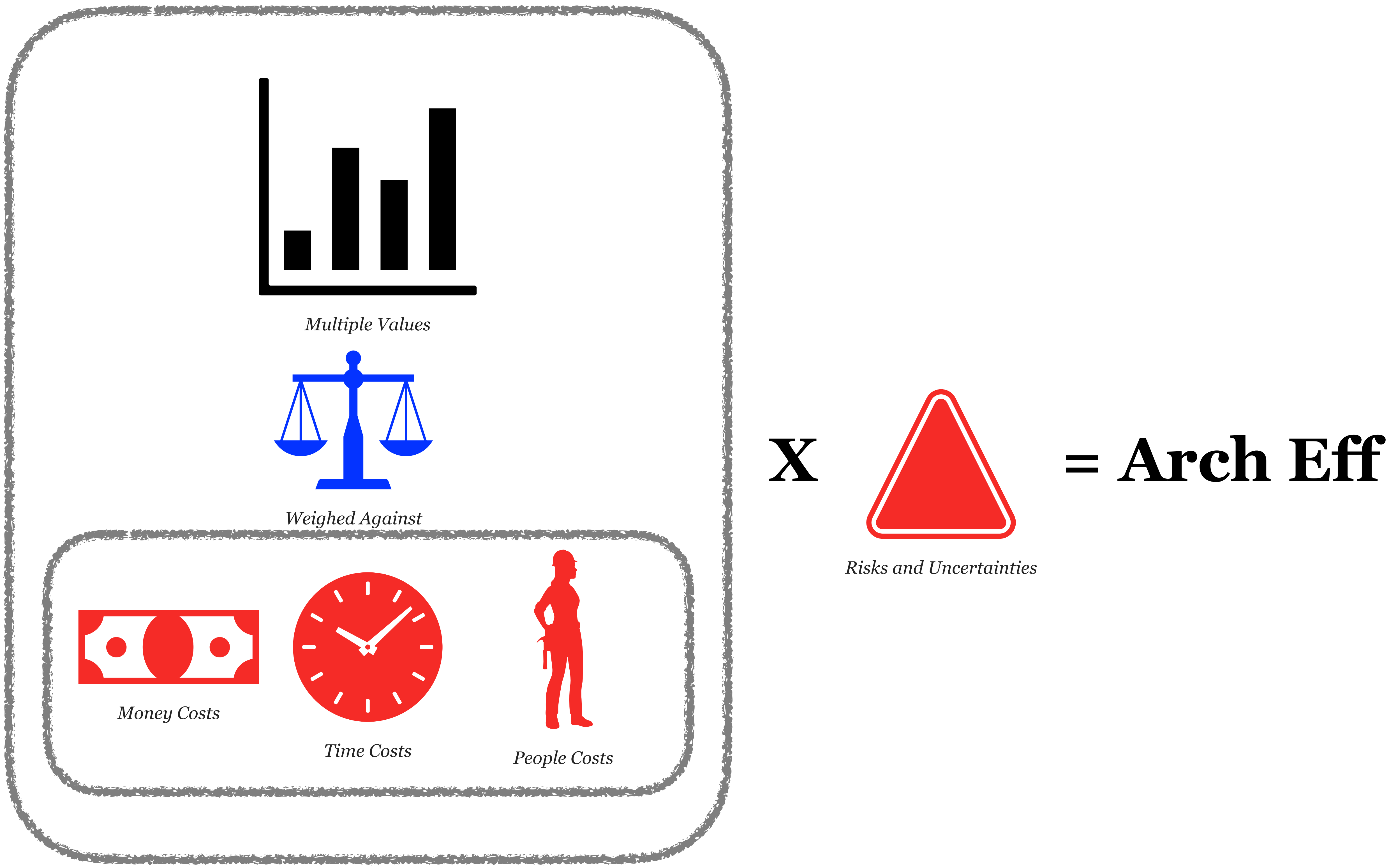
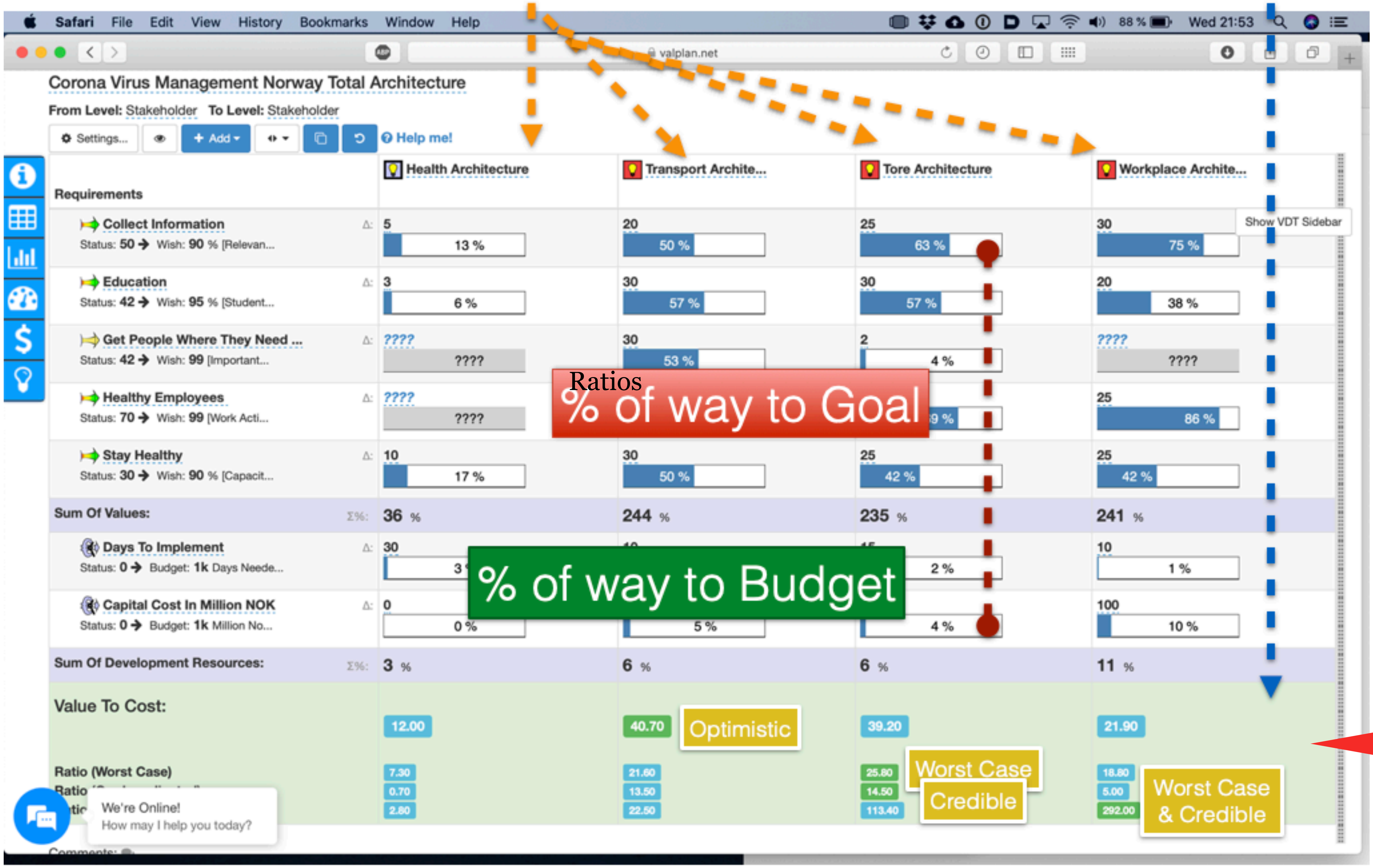


Figure 2.1 Multiple Value attributes of an architecture / Multiple cost attributes x Risks = Architecture Efficiency

2. Architecture Efficiency. ∑V/∑C x Risk. Table of AE

Corona Virus Planning: a 4-team class 2020,

The IE Table shows 4 Solutions x 6 Attributes + 4 £/V



Corona Virus Management Norway Total Architecture

From Level: Stakeholder To Level: Stakeholder

Requirements	Health Architecture	Transport Archite...	Tore Architecture	Workplace Archite...
Collect Information Status: 50 → Wish: 90 % [Relevan...	5 13 %	20 50 %	25 63 %	30 75 %
Education Status: 42 → Wish: 95 % [Student...	3 6 %	30 57 %	30 57 %	20 38 %
Get People Where They Need ... Status: 42 → Wish: 99 [Important...	???? ????	30 53 %	2 4 %	???? ????
Healthy Employees Status: 70 → Wish: 99 [Work Acti...	???? ????	[illegible]	[illegible] 9 %	25 86 %
Stay Healthy Status: 30 → Wish: 90 % [Capacit...	10 17 %	30 50 %	25 42 %	25 42 %
Sum Of Values: Σ%:	36 %	244 %	235 %	241 %
Days To Implement Status: 0 → Budget: 1k Days Neede...	30 3	10	15 2 %	10 1 %
Capital Cost In Million NOK Status: 0 → Budget: 1k Million No...	0 0 %	[illegible] 5 %	[illegible] 4 %	100 10 %
Sum Of Development Resources: Σ%:	3 %	6 %	6 %	11 %
Value To Cost:	12.00	40.70	39.20	21.90
Ratio (Worst Case)	7.30	21.60	25.80	18.80
Ratio [illegible]	0.70	13.50	14.50	5.00
[illegible]	2.80	22.50	113.40	292.00

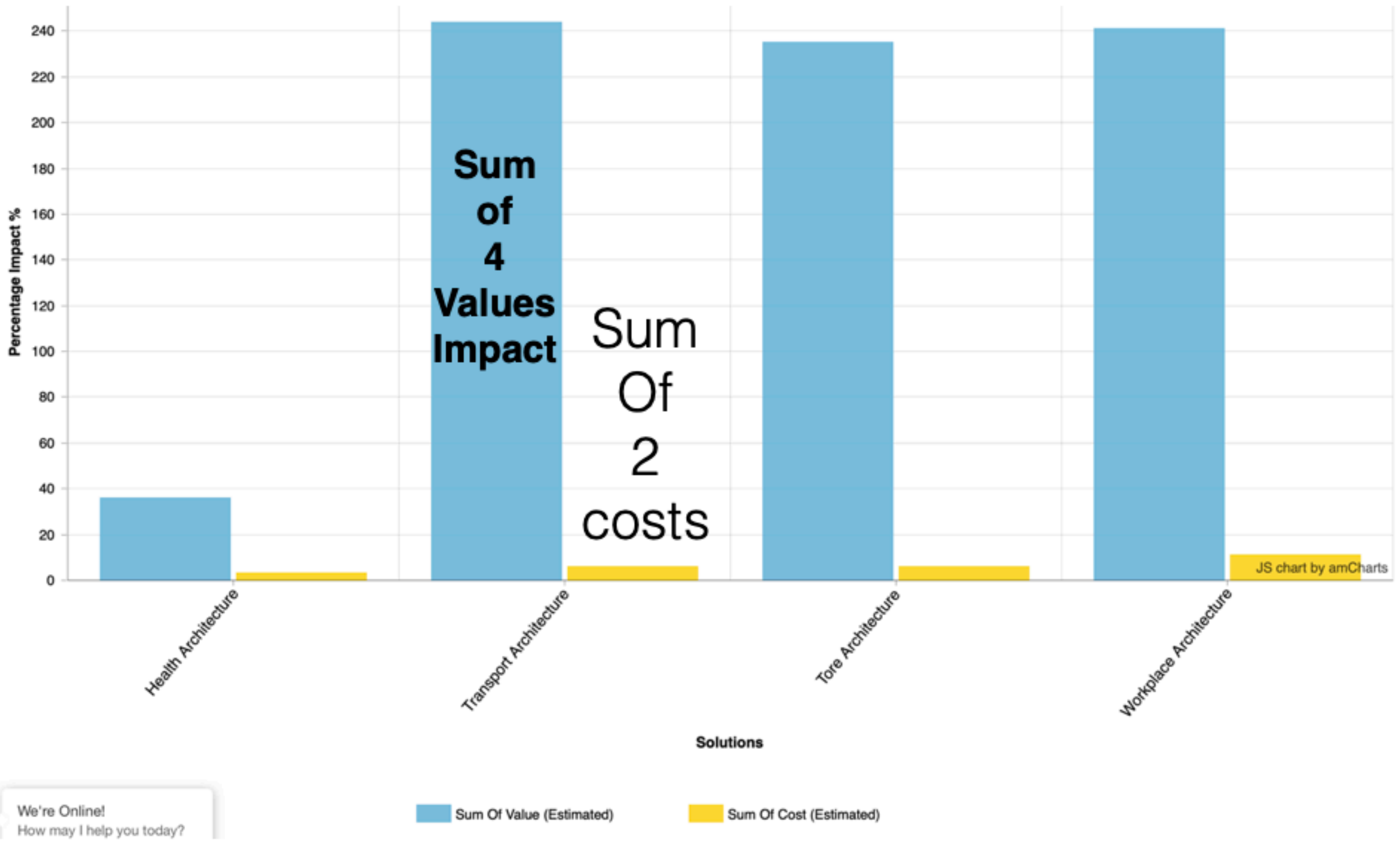
Efficiency Ratios x Risk

X

Figure 2.2 A ValPlan Impact Estimation Table (IET) adding up the values for each 4 sub-architectures, and adding up the costs, and then modifying the V/C by 3 different types of risk and uncertainty (see Part 12 for details).

The actual risk numbers are not visible here (quite noisy) , but if we choose to display them, or drip down into cell, they are there to analyze or audit.

The 'Complex' Table, simplified
Which solution is best 'values for costs'?



X

Figure 2.3 The table data on previous page (Fig 2.2) Summarized as a Bar Chart. Without any adjustment for Risks. It is a tight competition for 3 of the architecture options. The risk factors, and using numbers, will finally decide for the 4th on (Workplace Architecture) at a big difference 3x better(292) than the other 3 (113, 22, 2.8) see bottom line of table above)..

3. ‘ Constraint Respect’ in Architecture

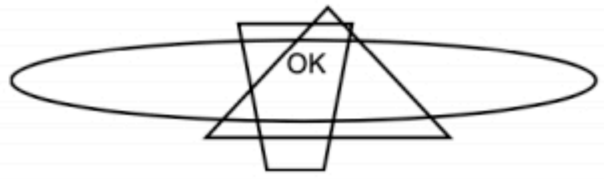


Figure G3
Constraints impose restrictions on both other requirements and designs. But the remaining space (‘OK’) gives considerable freedom to set more-exact requirements, and to specify more-exact designs.

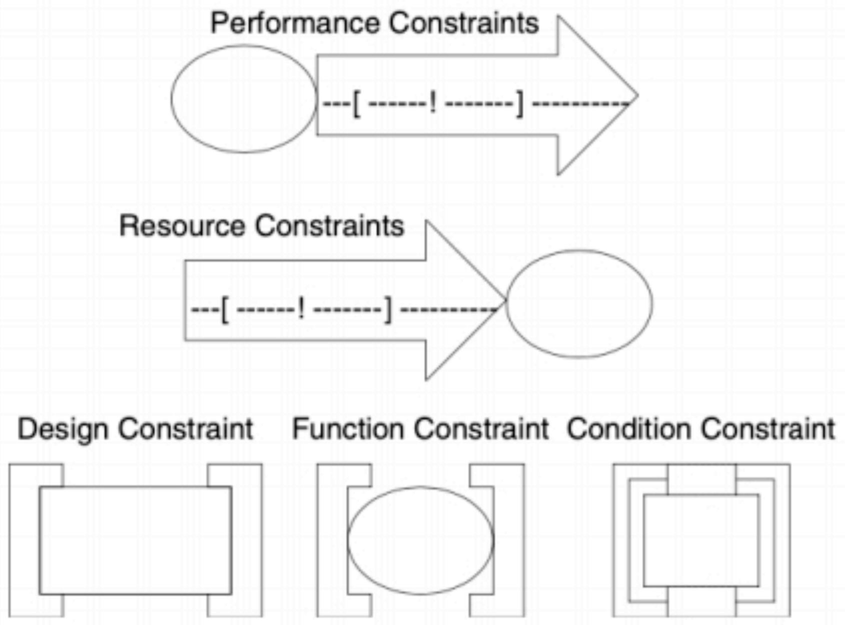


Figure G4
Drawn Icons for Constraints.

Copied from Figure 1.4 A and B, Source [CE]

Constraints [Part 1.4] were initially presented as a type of requirements, coming from stakeholders..

We can also see that constraints are of several different types: in particular binary constraints, and scalar constraints.

The question here is how the constraints impact our architecture process, and decision making.

Constraints eliminate architecture options

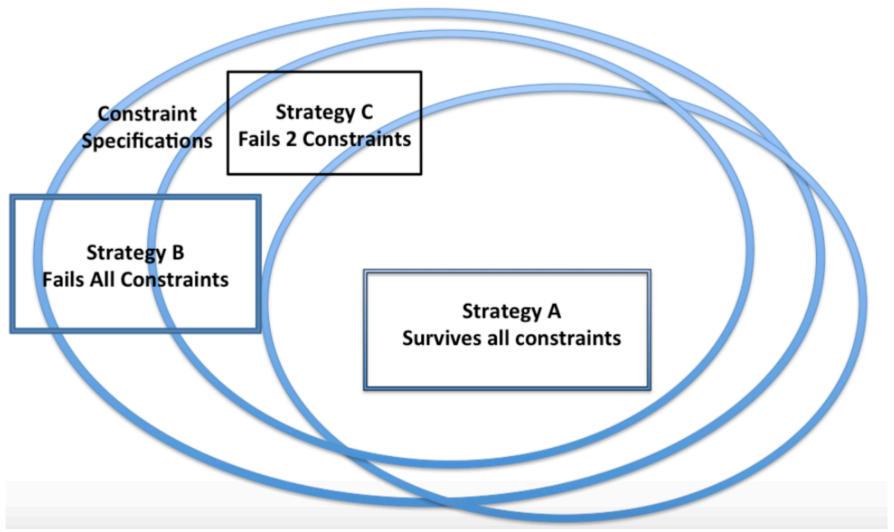

Constraint Specifications
Strategy C Fails 2 Constraints
Strategy B Fails All Constraints
Strategy A Survives all constraints

Figure 1.4C From [VP Chapter 3]

3. Architecture Constraint Principles © Gilb 2020

1. **RESTRICTION INTENDED**: Constraints are stakeholder requirements which intend to restrict your architecture choices options, for stakeholder reasons.

2. **CONSTRAINT PRIORITY**: Stakeholders are not all powerful, they have different levels of power, and consequent priority to impose constraint requirements; so constraints might be dropped, waived or modified to satisfy higher-priority stakeholders, and to satisfy higher-priority requirements (values and resources).

3. **QUESTION THEM**: Stakeholders may have 'changed their minds' or 'never really intended to require a constraint', so when constraints are blocking better architecture, the architect needs to question the constraints, never blindly accept them.

4. **REAL CONSTRAINT**: The architect can probe to find the underlying justification or reason for constraints, from the stakeholder point of view: and can potentially find an architecture-specification way, to satisfy the stakeholder's *real* needs, while ignoring or modifying the constraint.

5. **NARROWER CONSTRAINTS**: Constraints may have been formulated *too widely in scope* and *time*, inadvertently. The architect can suggest narrower constraint formations, which still fully-satisfy the stakeholders, but which then allow the architect greater ability to find better architecture, to satisfy the other requirements.

6. **ADJUST LEVELS**: Scalar constraints might have arbitrary but useful levels, which nobody thought would stop good architecture, at the time they were stated: so the architect should feel free to question them, and to respecify levels, and conditions; and get approval for the requirement-change, from appropriate stakeholders.

7. **FUZZY CONSTRAINTS**: the terms used in the constraint specification are very likely ambiguous. They should ideally be unambiguous of course, but people are undisciplined. So, if an unfavorable interpretation, is stopping better architecture, then you can try to make, and get-approved, a clearer interpretation.

8. **ARCH TEST**: In theory, the architecture process requires all specified, and all changed architecture specifications to be subjected to a *constraints test*: does the architecture spec violate or threaten to violate any constraint. You will need a clear requirements checklist of all constraints to do this, and it is a *clear responsibility* for the *responsible architect* to check. The check should be *noted, with results, directly in the architecture specification object*. The 'Test' parameter is suitable.

9. **REVIEW**: the same process of checking architecture against constraints should be done by independent (of the architect) reviewers. Using the rule-based Spec QC process [CE, P5].

10. **UPDATES**: all changes, however seemingly slight, need to be checked by the responsible-change-owner architect against all current constraints. And such checks should be noted in the architecture specification object (Test: Parameter)

These 'Arch Constraint' Principles were originated by Tom Gilb 030920 for SEA book

3. Specific methods for Constraint testing against

Scalar Constraints Testing

Let us use a simple example:

———— Here are the Scalar Constraint Requirements——

Constraint Cold: Tolerable: 14 Deg C

Constraint Hot: Tolerable 39.9 Deg C

——- Here is an Architecture Specification, and Notation—

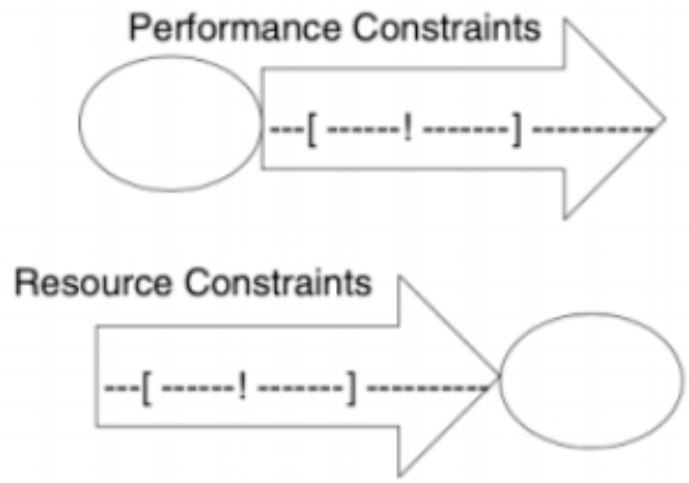
Heater: a thermostatically controlled app and lower C limits electric heater.

Condition: the limits are set at the constraint levels, for now Turn On Heater under 14C, and Turn Off at 40C.

Issue: if the outside temperature itself causes room temperature to exceed 39,9 Constraint, the Heater has no help, alone.

Mitigation: Include some kind of Cooling device to kick in, or insulation.

Responsible: Building Environment Architect.



3. Specific methods for Constraint testing against architecture:Design Constraint

A design constraint is interesting, because it is explicit about architecture itself.

Design Constraint: A requirement specification, that demands, or forbids, something regarding a design. (from Glossary at end of this book)

Examples:

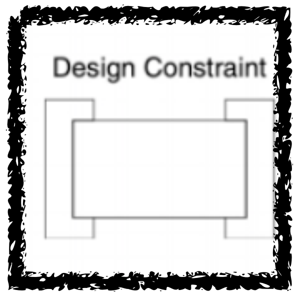
Focus Constraint: **Architecture Constraint**: The architecture must all be clearly focussed on delivering the critical stakeholder value levels as early as possible.

Exclusion of Architecture: **Architecture Constraint**: no architecture component shall originate from, or be in any way dependent on, or controlled by Nations on our list of Aggressor Nations, or Dictator Nations. The first priority will be sourcing from Friendly Nations List or our own Nation and close trade and military allies.

All Constraints shoud be *annotated* with **sources and justifications**, to help us deal with them.

The same architecture processes as listed above apply (Principles 8, 9, 10).

1. The Design Constraints must be clearly labeled with correct type (Design Constraint, or Architecture Constraint).
2. The architect is responsible in the first instance for being aware of the current Constraint lists, checking them when any new architecture component is suggested, and specifying, in direct connection with the Architecture Component specification.
3. The Architect is responsible for:
    1. Noting negative vetting (no known constraints violated), or
    2. doubt (Possible violation of Architecture Constraint XYZ, if xxx) , and
    3. Positive Vetting (unacceptable at the moment because of the following Architecture Constraint:XXX.



4. The Architect is at liberty to take constructive action to ignore, or change the Constraint, with reasons or permissions given.
5. A Specification QC (review using Rules for Architecture Specs [B1, P7]. Can quantify the defect (Rule Violation) Density, and allow Exit of the Architecture from the process, or refuse Exit.
6. When review is exited, this fact needs to be annotated on the version of the Architecture spec, or Arch. Spec. Object concerned.

4. Architecture Decomposition

From our Glossary

Decomposition:(+030920) refers to a process of decomposing into more-detailed sub-components, such as Sub-architectures, and Sub-Values, any architecture specification objects (including functions, values, resources, architecture, constraints, time) so as to obtain smaller specification-objects, for any purpose; such as early delivery, optimization, separation of concerns (like suppliers), & managing risks. See Part 4 here, and [B2.**Decomposition**: https://tinyurl.com/VPDecomposition].

mid 18th century (in the sense 'separate into simpler constituents'): from French ***décomposer***, from ***de-*** (expressing reversal) + ***composer***.

MATHEMATICS
express (a number or function) as a combination of simpler components.
"in how many ways can one decompose a number as a sum of squares?"

Note: I am aware of the dictionary definition referring to rotting etc. but I will stick to the De-compose in the **dictionary sense** of breaking into separate parts or components. rather than using terms like 'break down", "split up" . I am of the 18th Century French opinion, Cartesian Decomposition.

Cartesian mosaic.

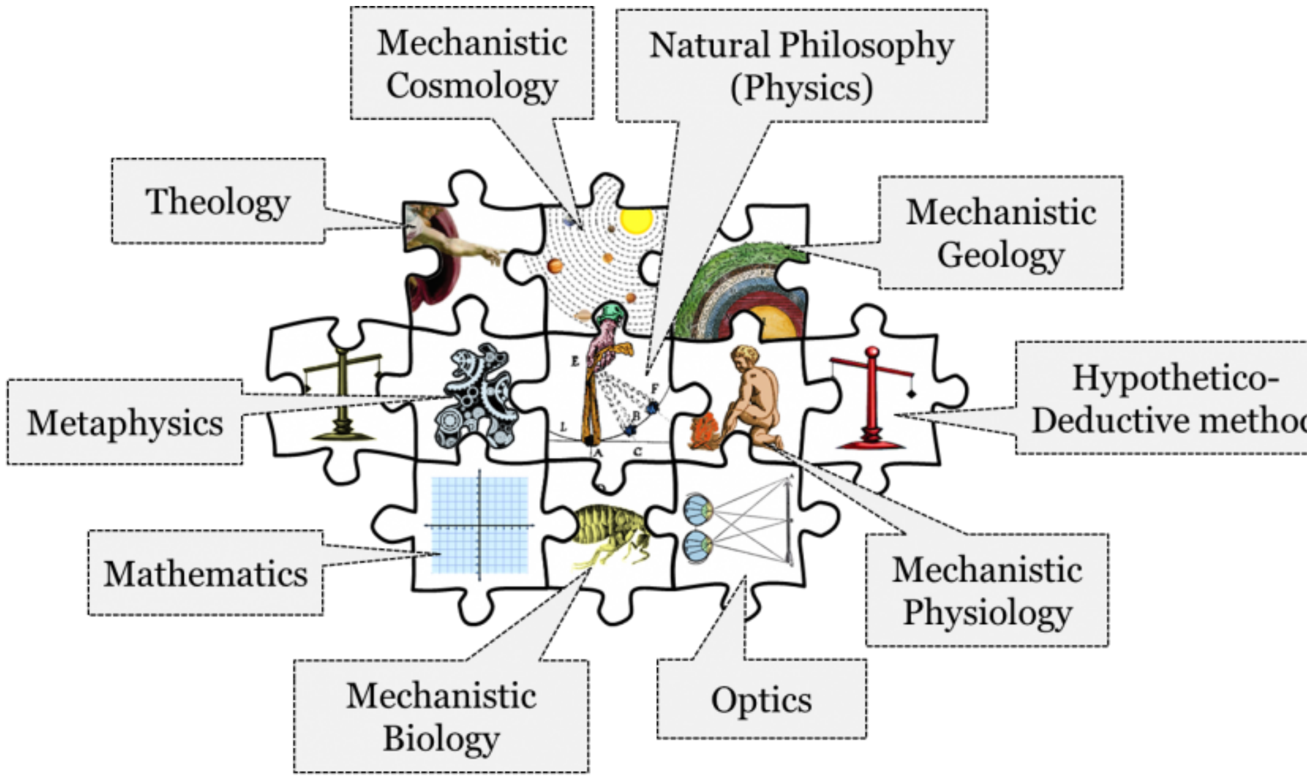
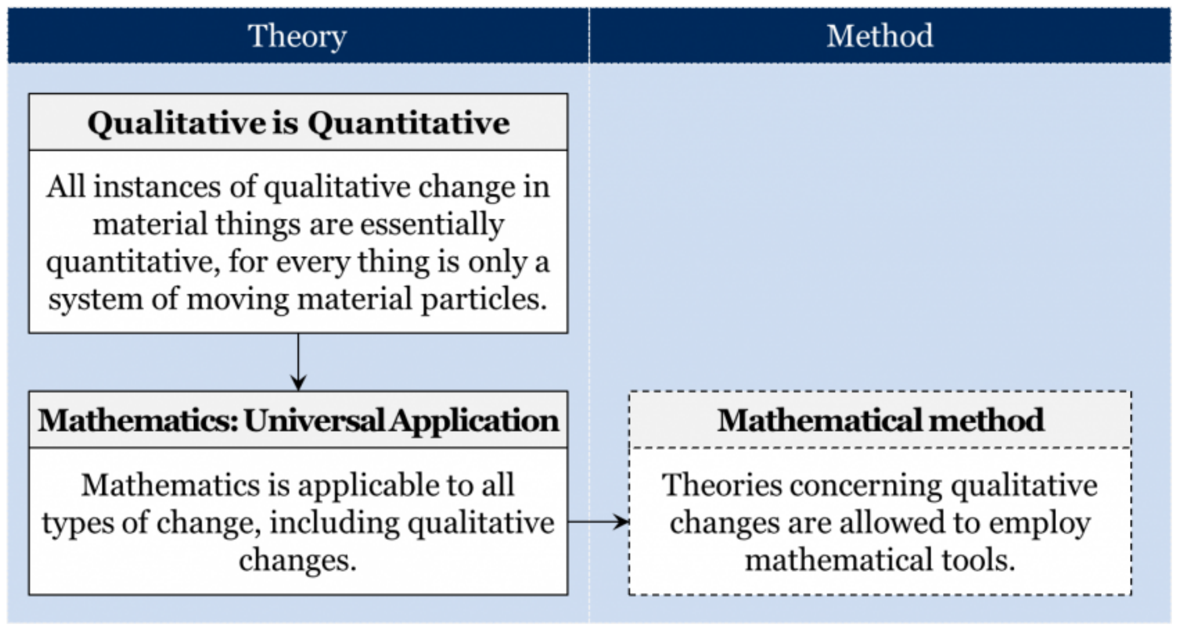


Figure 4.0 A. http://hakobsandbox.openetext.utoronto.ca/part/chapter-8/

I love this! Qualitative is Quantitative. This is my hero Rene Descartes.



Qualitative is Quantitative

All instances of qualitative change in material things are essentially quantitative, for every thing is only a system of moving material particles.

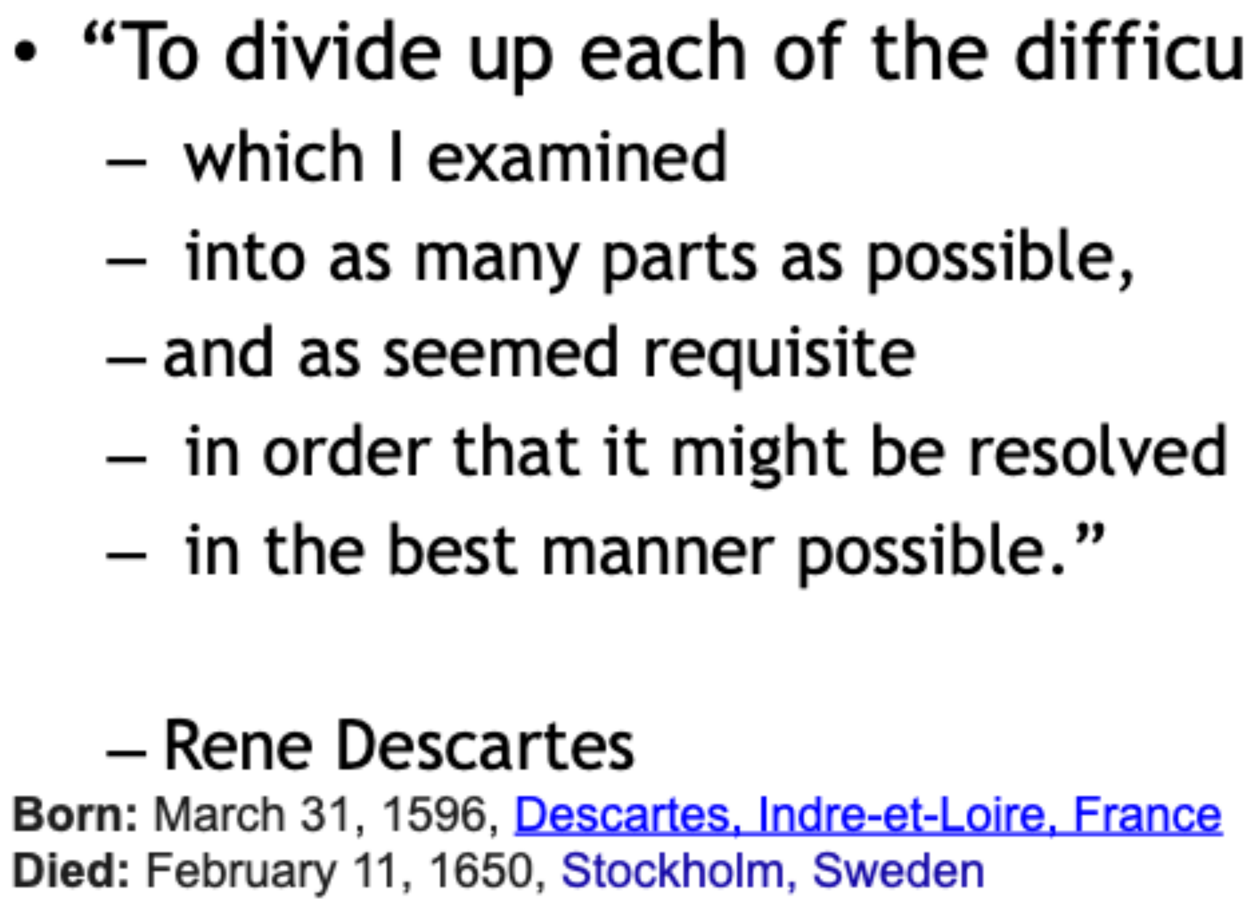
Mathematics: Universal Application

Mathematics is applicable to all types of change, including qualitative changes.

Mathematical method

Theories concerning qualitative changes are allowed to employ mathematical tools.

Figure 4.0 B. http://hakobsandbox.openetext.utoronto.ca/part/chapter-8/



- "To divide up each of the difficulties
  - which I examined
  - into as many parts as possible,
  - and as seemed requisite
  - in order that it might be resolved
  - in the best manner possible."

  - Rene Descartes

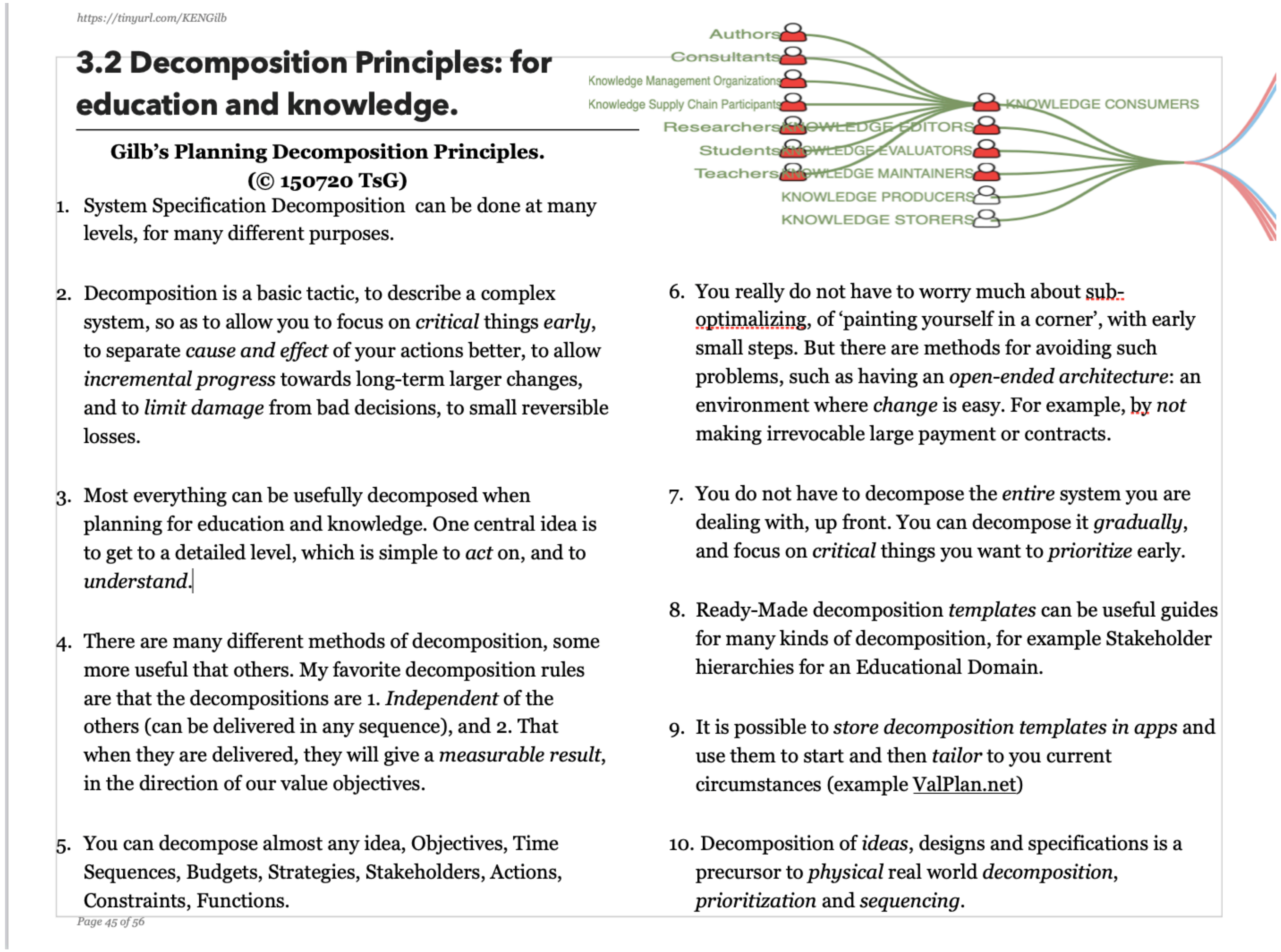
Born: March 31, 1596, Descartes, Indre-et-Loire, France
Died: February 11, 1650, Stockholm, Sweden

Figure 4.0 C Nice to see the original French wording. ***"décomposition"***

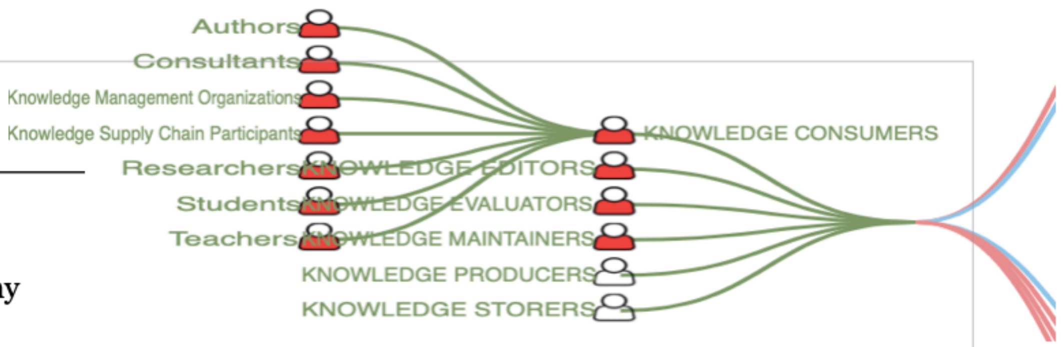
la règle de décomposition : « Le second, de diviser chacune des difficultés que j'examinerais en autant de parcelles qu'il se pourrait et qu'il serait requis pour les mieux résoudre. » ;

Source: https://1000-idees-de-culture-generale.fr/discours-de-la-methode/

Title



3.2 Decomposition Principles: for education and knowledge.



Gilb's Planning Decomposition Principles.
(© 150720 TsG)

1. System Specification Decomposition can be done at many levels, for many different purposes.

2. Decomposition is a basic tactic, to describe a complex system, so as to allow you to focus on *critical* things *early*, to separate *cause and effect* of your actions better, to allow *incremental progress* towards long-term larger changes, and to *limit damage* from bad decisions, to small reversible losses.

3. Most everything can be usefully decomposed when planning for education and knowledge. One central idea is to get to a detailed level, which is simple to *act* on, and to *understand*.

4. There are many different methods of decomposition, some more useful that others. My favorite decomposition rules are that the decompositions are 1. *Independent* of the others (can be delivered in any sequence), and 2. That when they are delivered, they will give a *measurable result*, in the direction of our value objectives.

5. You can decompose almost any idea, Objectives, Time Sequences, Budgets, Strategies, Stakeholders, Actions, Constraints, Functions.

6. You really do not have to worry much about sub-optimalizing, of 'painting yourself in a corner', with early small steps. But there are methods for avoiding such problems, such as having an *open-ended architecture*: an environment where *change* is easy. For example, by *not* making irrevocable large payment or contracts.

7. You do not have to decompose the *entire* system you are dealing with, up front. You can decompose it *gradually*, and focus on *critical* things you want to *prioritize* early.

8. Ready-Made decomposition *templates* can be useful guides for many kinds of decomposition, for example Stakeholder hierarchies for an Educational Domain.

9. It is possible to *store decomposition templates in apps* and use them to start and then *tailor* to you current circumstances (example ValPlan.net)

10. Decomposition of *ideas*, designs and specifications is a precursor to *physical* real world *decomposition*, *prioritization* and *sequencing*.

Figure 4.1A. Source tiny url.com/KENGilb. The Knowledge book.

3.2 Decomposition Principles: for education and knowledge.

1. System Specification Decomposition can be done at many levels, for many different purposes.

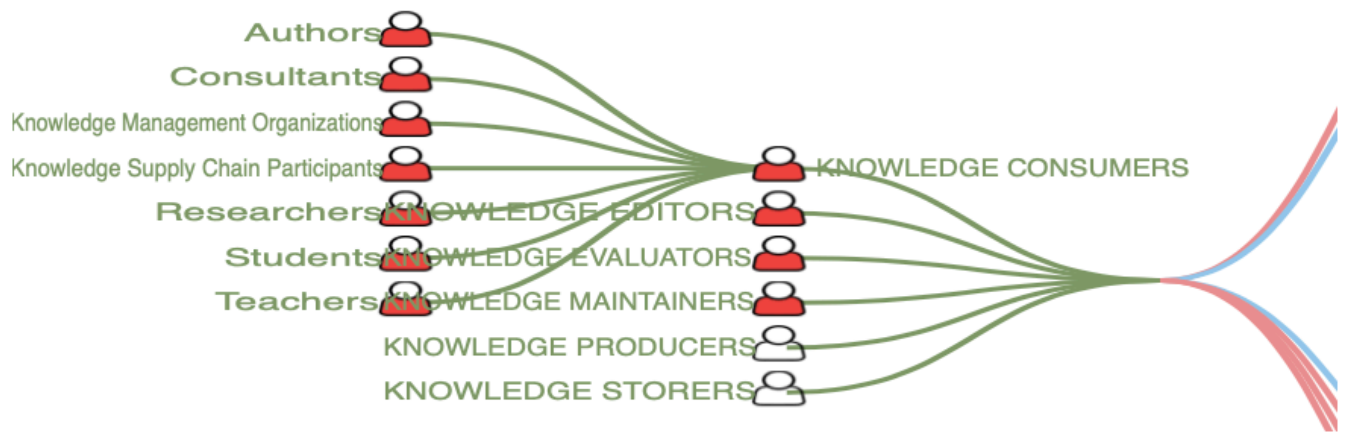
2. Decomposition is a basic tactic, to describe a complex system, so as to allow you to focus on *critical* things *early*, to separate *cause and effect* of your actions better, to allow *incremental progress* towards long-term larger changes, and to *limit damage* from bad decisions, to small reversible losses.

3. Most everything can be usefully decomposed when planning for education and knowledge. One central idea is to get to a detailed level, which is simple to *act* on, and to *understand*.

4. There are many different methods of decomposition, some more useful that others. My favorite decomposition rules are that the decompositions are 1. *Independent* of the others (can be delivered in any sequence), and 2. That when they are delivered, they will give a *measurable result*, in the direction of our value objectives.

5. You can decompose almost any idea, Objectives, Time Sequences, Budgets, Strategies, Stakeholders, Actions, Constraints, Functions.

Figure 3.2.5 Stakeholder hierarchy. Knowledge `consumers.



6. You really do not have to worry much about sub-optimalizing, of 'painting yourself in a corner', with early small steps. But there are methods for avoiding such problems, such as having an *open-ended architecture*: an environment where *change* is easy. For example, by *not* making irrevocable large payment or contracts.

7. You do not have to decompose the *entire* system you are dealing with, up front. You can decompose it *gradually*, and focus on *critical* things you want to *prioritize* early.

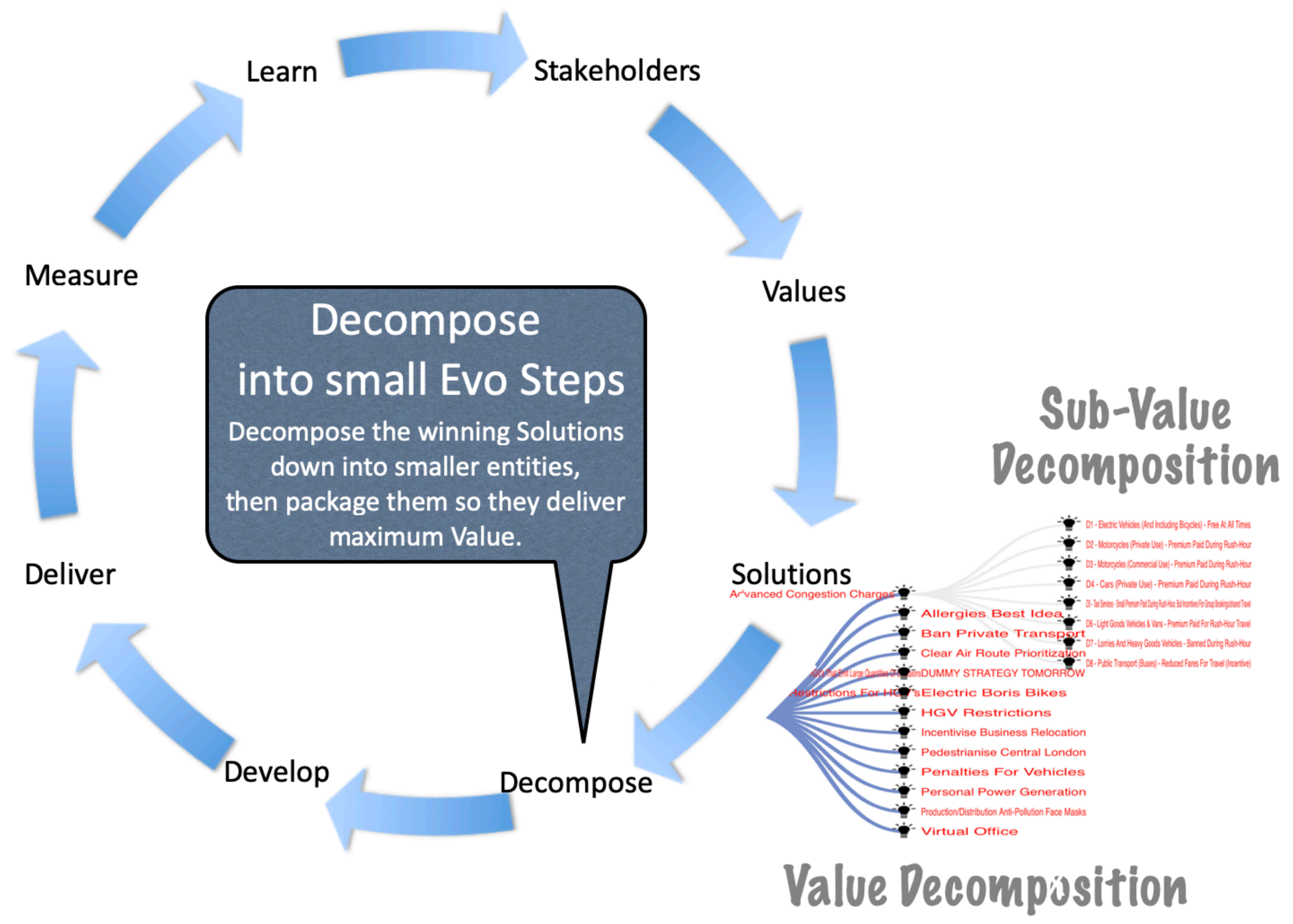
8. Ready-Made decomposition *templates* can be useful guides for many kinds of decomposition, for example Stakeholder hierarchies for an Educational Domain.

9. It is possible to *store decomposition templates in apps* and use them to start and then *tailor* to your current circumstances (example ValPlan.net)

10. Decomposition of *ideas*, designs and specifications is a precursor to *physical* real world *decomposition*, *prioritization* and *sequencing*.

4.1 Architecture Decomposition

Decomposition of architecture, and of value requirements; is a regular agile incremental activity



Learn
Stakeholders
Values
Solutions
Decompose
Develop
Deliver
Measure
Decompose into small Evo Steps
Decompose the winning Solutions down into smaller entities, then package them so they deliver maximum Value.
Sub-Value Decomposition
Advanced Congestion Charges
Allergies Best Idea
Ban Private Transport
Clear Air Route Prioritization
HGV's That Emit Large Quantities Of Emissions
DUMMY STRATEGY TOMORROW
Restrictions For HGV's
Electric Boris Bikes
HGV Restrictions
Incentivise Business Relocation
Pedestrianise Central London
Penalties For Vehicles
Personal Power Generation
Production/Distribution Anti-Pollution Face Masks
Virtual Office
D1 - Electric Vehicles (And Including Bicycles) - Free At All Times
D2 - Motorcycles (Private Use) - Premium Paid During Rush-Hour
D3 - Motorcycles (Commercial Use) - Premium Paid During Rush-Hour
D4 - Cars (Private Use) - Premium Paid During Rush-Hour
D5 - Taxi Services - Small Premium Paid During Rush-Hour, But Incentives For Group Bookings/shared Travel
D6 - Light Goods Vehicles & Vans - Premium Paid For Rush-Hour Travel
D7 - Lorries And Heavy Goods Vehicles - Banned During Rush-Hour
D8 - Public Transport (Buses) - Reduced Fares For Travel (Incentive)
Value Decomposition

Figure 4.1 B
Source Kai Gilb Evo Cycle slides + ValPlan Decomposition Example BCS Workshop London Congestion

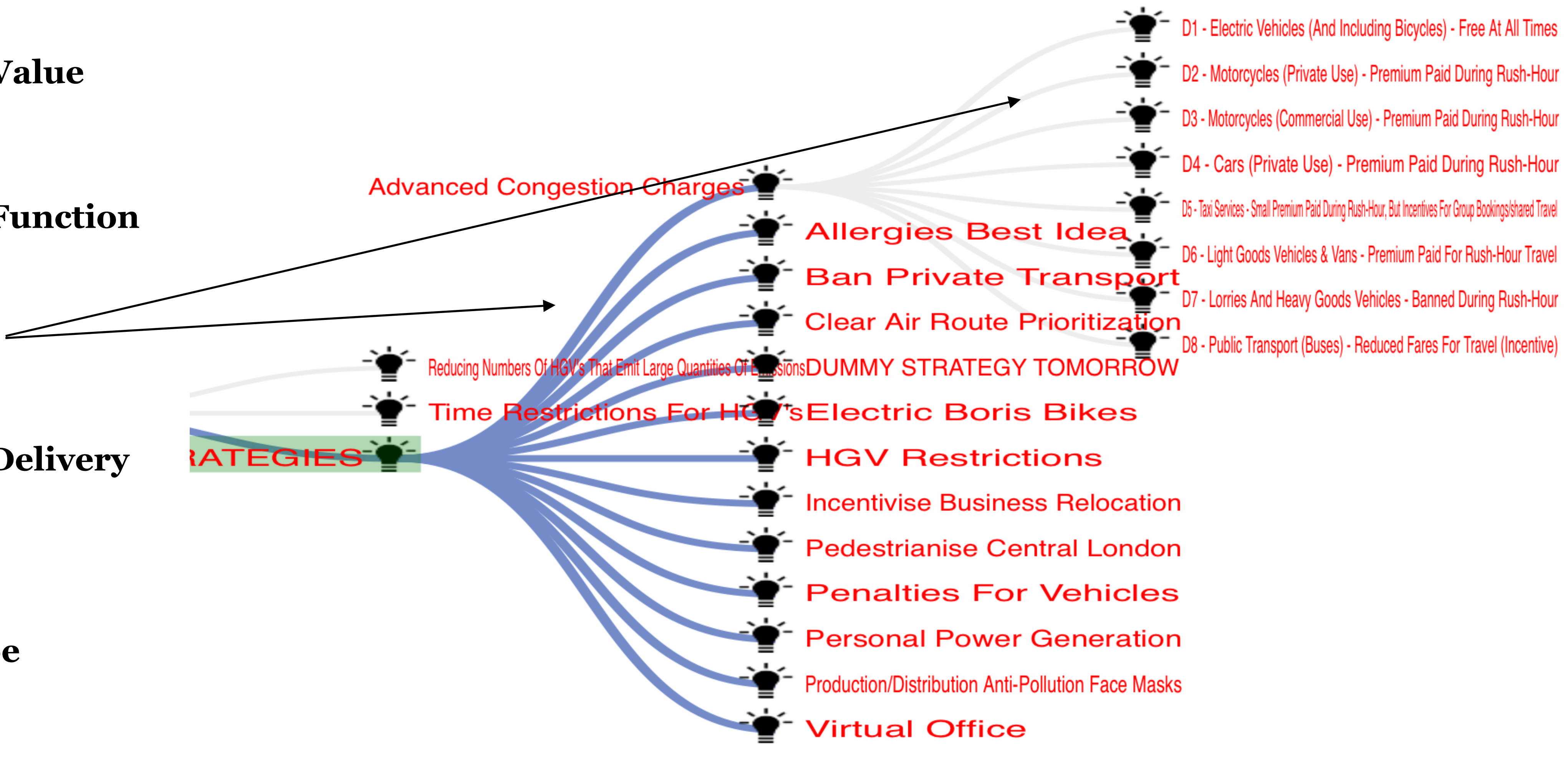
For 51 pages of detail on my decomposition principles I recommend the Decomposition Chapter of 'Value Planning' [free, B2.**Decomposition**: https://tinyurl.com/VPDecomposition]

4.1 Architecture Decomposition : Some basic decomposition Tactics:

Here are some basic decomposition tactics the architect can apply.

1. **Decomposition of Value Requirements**
2. **Decomposition of Function**
3. **Decomposition of Architecture**
4. **Decomposition of Delivery Time**
5. **Decomposition of Conditions or Scope**
6. **Decomposition by Stakeholder**

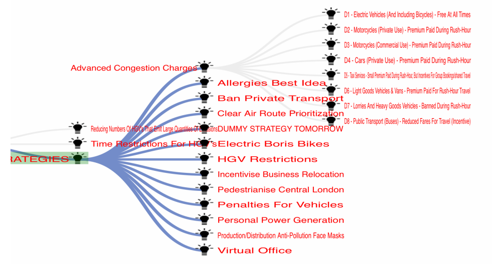
Two levels of Architecture Decomposition, captured digitally in ValPlan as a diagram display.
I instructed the 'Advanced Congestion Charges' team to divide up into D1 to D8
And make sure each on delivered value, and was independently implementable.



D1 - Electric Vehicles (And Including Bicycles) - Free At All Times
D2 - Motorcycles (Private Use) - Premium Paid During Rush-Hour
D3 - Motorcycles (Commercial Use) - Premium Paid During Rush-Hour
D4 - Cars (Private Use) - Premium Paid During Rush-Hour
D5 - Taxi Services - Small Premium Paid During Rush-Hour, But Incentives For Group Bookings/shared Travel
D6 - Light Goods Vehicles & Vans - Premium Paid For Rush-Hour Travel
D7 - Lorries And Heavy Goods Vehicles - Banned During Rush-Hour
D8 - Public Transport (Buses) - Reduced Fares For Travel (Incentive)
Advanced Congestion Charges
Allergies Best Idea
Ban Private Transport
Clear Air Route Prioritization
Reducing Numbers Of HGV's That Emit Large Quantities Of Emissions
DUMMY STRATEGY TOMORROW
Time Restrictions For HGV's
Electric Boris Bikes
RATEGIES
HGV Restrictions
Incentivise Business Relocation
Pedestrianise Central London
Penalties For Vehicles
Personal Power Generation
Production/Distribution Anti-Pollution Face Masks
Virtual Office

Figure 4.1 C *Source BCS Workshop, London Congestion Planning*

4.1 Architecture Decomposition : Six basic decomposition Tactics: Some detail.


D1 - Electric Vehicles (And Including Bicycles) - Free At All Times
D2 - Motorcycles (Private Use) - Premium Paid During Rush-Hour
D3 - Motorcycles (Commercial Use) - Premium Paid During Rush-Hour
D4 - Cars (Private Use) - Premium Paid During Rush-Hour
Advanced Congestion Charges
D5 - Taxi Services - Small Premium Paid During Rush-Hour, But Incentives For Group Bookings/shared Travel
Allergies Best Idea
D6 - Light Goods Vehicles & Vans - Premium Paid For Rush-Hour Travel
Ban Private Transport
D7 - Lorries And Heavy Goods Vehicles - Banned During Rush-Hour
Clear Air Route Prioritization
D8 - Public Transport (Buses) - Reduced Fares For Travel (Incentive)
Reducing Numbers Of HGV's That Emit Large Quantities Of Emissions
DUMMY STRATEGY TOMORROW
Time Restrictions For HGV's
Electric Boris Bikes
RATEGIES
HGV Restrictions
Incentivise Business Relocation
Pedestrianise Central London
Penalties For Vehicles
Personal Power Generation
Production/Distribution Anti-Pollution Face Masks
Virtual Office

Here
are some basic <u>decomposition tactics</u> the architect can apply.

1. **Decomposition of Value Requirements, By Value and By level, then Cost Optimization**

a. If you have 10 Critical Stakeholder Values for a project, then your Values are already 'decomposed' into 10 different value-agendas.

b. You can choose any one value requirement to start improving, towards their ***constraint*** levels (Tolerable). First priority is survival.

c. When all 10 in turn, have reached Constraint levels, you can turn your attention to selecting one, and delivering the **Target** levels (Goal), until all 10 Goal levels (or all Target levels) are reached. You now have a formal 'project **Success**' (all Targets reached).

d. You can now turn your attention towards any specified **Stretch** levels, if you have enough resources to do so. If you have resources remaining, to do the work of resource optimization, or if you can 'rgue for getting such resources' in addition.

e. You can also attempt *resource optimization*: which means - reducing costs of having reached the Target Value levels. Example reducing future Technical Debt.

2. **Decomposition of Function**

a. You can decompose major function areas into sub-functions

b. And then prioritize some functions for earlier delivery

3. **Decomposition of Architecture**

a. Decompose major top-level architecture-ideas into smaller (10%) sub-architectures, which fulfill the following conditions:

b. They will, when delivered to the real living system (being incremented towards target value levels), **actually deliver** some **value increment** towards at least one, planned-value target.

c. They are **independent of each other**, meaning they can be implemented in any prioritized sequence, without any of the others being in place yet. Prioritization freedom.

d. If the '10% decompositions' are larger than the desired value-delivery increment, then continue decomposition, usually until the increment is about 2% of total resources, or a *week to a month* of calendar time. Get small increments.

4. **Decomposition of Delivery Time.**

This sort of time decomposition is often already in place, using various Value statement deadlines, and various condition combinations. But to the degree it is *not* in place, we can (the architect can) decompose, for example a one-year delivery task, into smaller 10%-or-less increments, of about a month each. Increasing the Value levels about 10% of the way towards a Target level, each month. Roughly is OK.

5. **Decomposition of Conditions or Scope**

a. By selecting prioritized sets of [Scale Parameter] Conditions you can decompose into Scope sets; and prioritize a stream of different Scopes. A Simple example: decompose by Cities in a Nation, and perhaps also by School types..

6. **Decomposition by Stakeholder**

a. Decompose into sets of stakeholders, to satisfy requirements for.

b. Decompose into sets of Stakeholder to deliver Values (and other requirements) to.

4.1 Architecture Decomposition : Reasons why we decompose

Why is architecture decomposition useful? And why should the *architect* do it, rather than an agile project manager/ Product Owner ?

Decomposition *Usefulness*

1. **VALUE STREAM:** Decomposition enables us to have an early, almost continuous value stream, of increased critical stakeholder-value, getting better, going towards targeted value levels, on time. This is essential agile, essentially avoiding big bang projects.

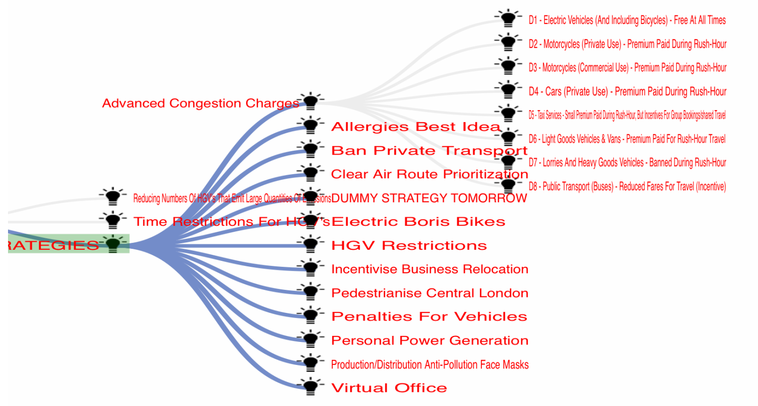
2. **SIMPLIFICATION:** Decomposition allows us to simplify large and complex systems, into a prioritized stream of small trials or experiments.

3. **EARLY:** Decomposition allows us to get extremely early results, *this financial cycle, this election period, before the 'coach' gets fired for a losing-streak incompetence.* <u>*Winning*</u> *streaks are so much nicer!*

4. **ARCH EFFORT SPREAD***:* the total architecture effort can be distributed throughout the project, not as a big-bang up front.

5. **FEEDBACK**: by delivering increments in small and limited packages, we can get several kinds of feedback (value, cost, human reactions) which if negative, gives us a fair chance, to adjust our architecture, or maybe to adjust our false expectations

6. **PARALLEL LEARNING**: parallel devlopment and implementation teams can learn from immediately-preceding implementations, and adjust their *own* delivery, architecturally, or in other ways (setting expectations, better followup staffing, etc.)



D1 - Electric Vehicles (And Including Bicycles) - Free At All Times
D2 - Motorcycles (Private Use) - Premium Paid During Rush-Hour
D3 - Motorcycles (Commercial Use) - Premium Paid During Rush-Hour
D4 - Cars (Private Use) - Premium Paid During Rush-Hour
D5 - Taxi Services - Small Premium Paid During Rush-Hour, But Incentives For Group Bookings/shared Travel
D6 - Light Goods Vehicles & Vans - Premium Paid For Rush-Hour Travel
D7 - Lorries And Heavy Goods Vehicles - Banned During Rush-Hour
D8 - Public Transport (Buses) - Reduced Fares For Travel (Incentive)
Advanced Congestion Charges
Allergies Best Idea
Ban Private Transport
Clear Air Route Prioritization
Reducing Numbers Of HGV's That Emit Large Quantities Of Emissions
DUMMY STRATEGY TOMORROW
Time Restrictions For HGV's
Electric Boris Bikes
RATEGIES
HGV Restrictions
Incentivise Business Relocation
Pedestrianise Central London
Penalties For Vehicles
Personal Power Generation
Production/Distribution Anti-Pollution Face Masks
Virtual Office

<u>Why</u> should the architect do decomposition *instead of* an agile 'project manager' or 'product owner' ?

1. Only the architect, with their complete architecture view, is capable of safe decomposition, and avoiding sub-optimization

2. The architect would lose control and responsibility, over the changes, and the whole effort. *A construction foreman cannot tell plumbers to change the plumbing from the blueprint, to save time.*

3. The 'project manager' would, in reality, have to have *architect training and* take *architect responsibility* for consequences.

4. The architect should not be just a front-end activity. They need to be directly and quantitatively engaged (like Quinnan in Cleanroom, [P3.3] at every incremental delivery, and every feedback agile-cycle, in order to be 'agile' in their response.

5. The Scrum agile product owner is not trained for any of this.

4.3 Architecture Decomposition , The ‘Polish Export’ Example

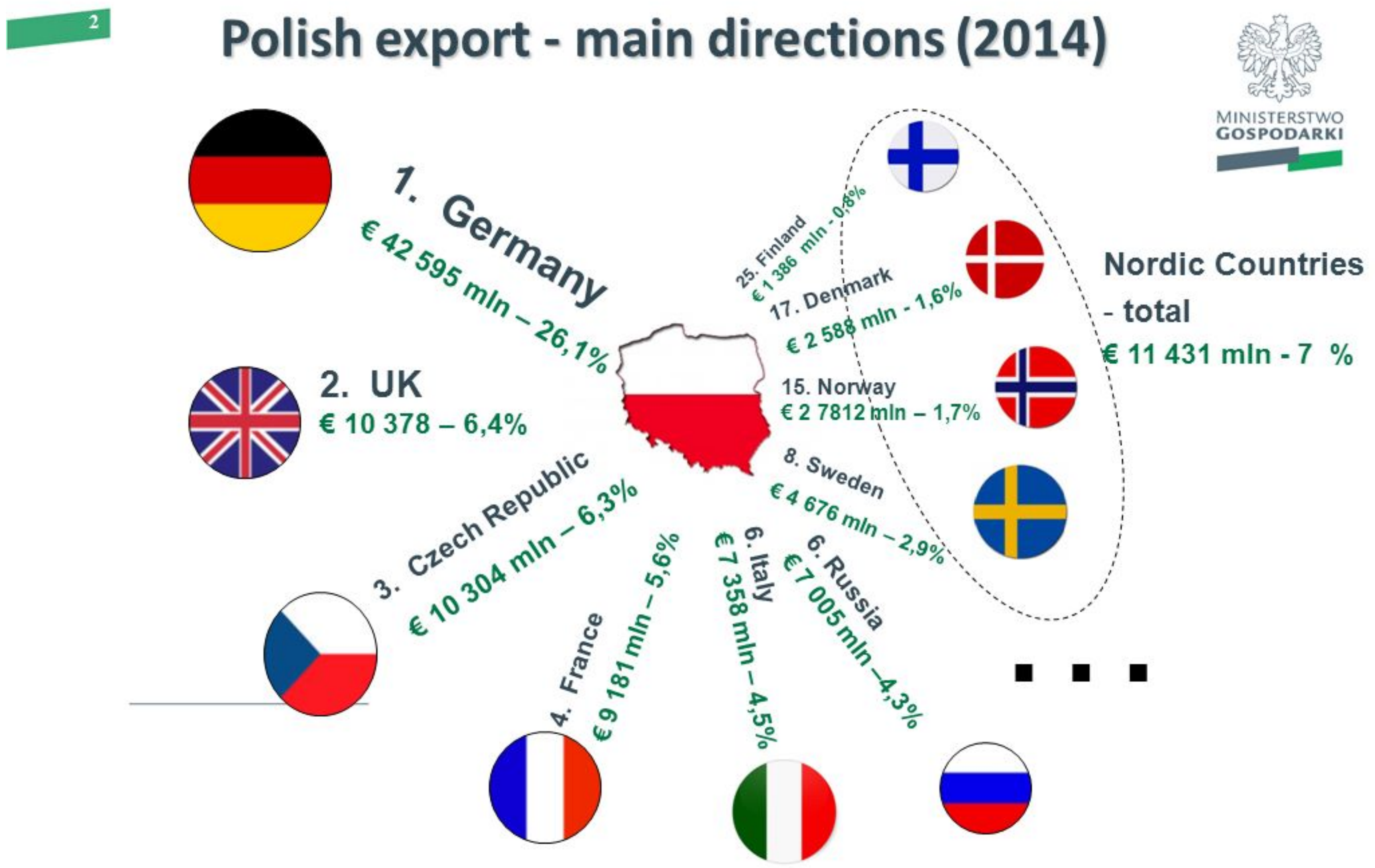


Figure 4.1 E. *https://images.slideplayer.com/38/10825192/slides/slide_2.jpg*
This exercise was from a two-day workshop in Warsaw, which I held, for a consultancy, and for a group of start-ups, under their care.
About 60 people attended. They worked in about 13 small groups in parallel. As you can see here, we used ValPlan to integrate their parallel work in real time. I could track progress, and even display it on a screen for all to see.

Day One we quantified their stakeholder values. Day Two we did the architecture, and late afternoon 2nd day we did impact estimation of the architecture, on their requirements.

This is my normal 'Architecture Engineering' Workshop.
These people were excellent workshop students. They dived in and did it right.
It may be ‘student’ work.
But the ‘engineering’ content is far superior to the primitive Enterprise Architecture practices I see internationally.
That is why I am writing this book. To 'reach the parts' that personal teaching cannot. Share a copy with the needy.

Decomposition into second level of architecture detail, independent value-delivery sub-architectures.

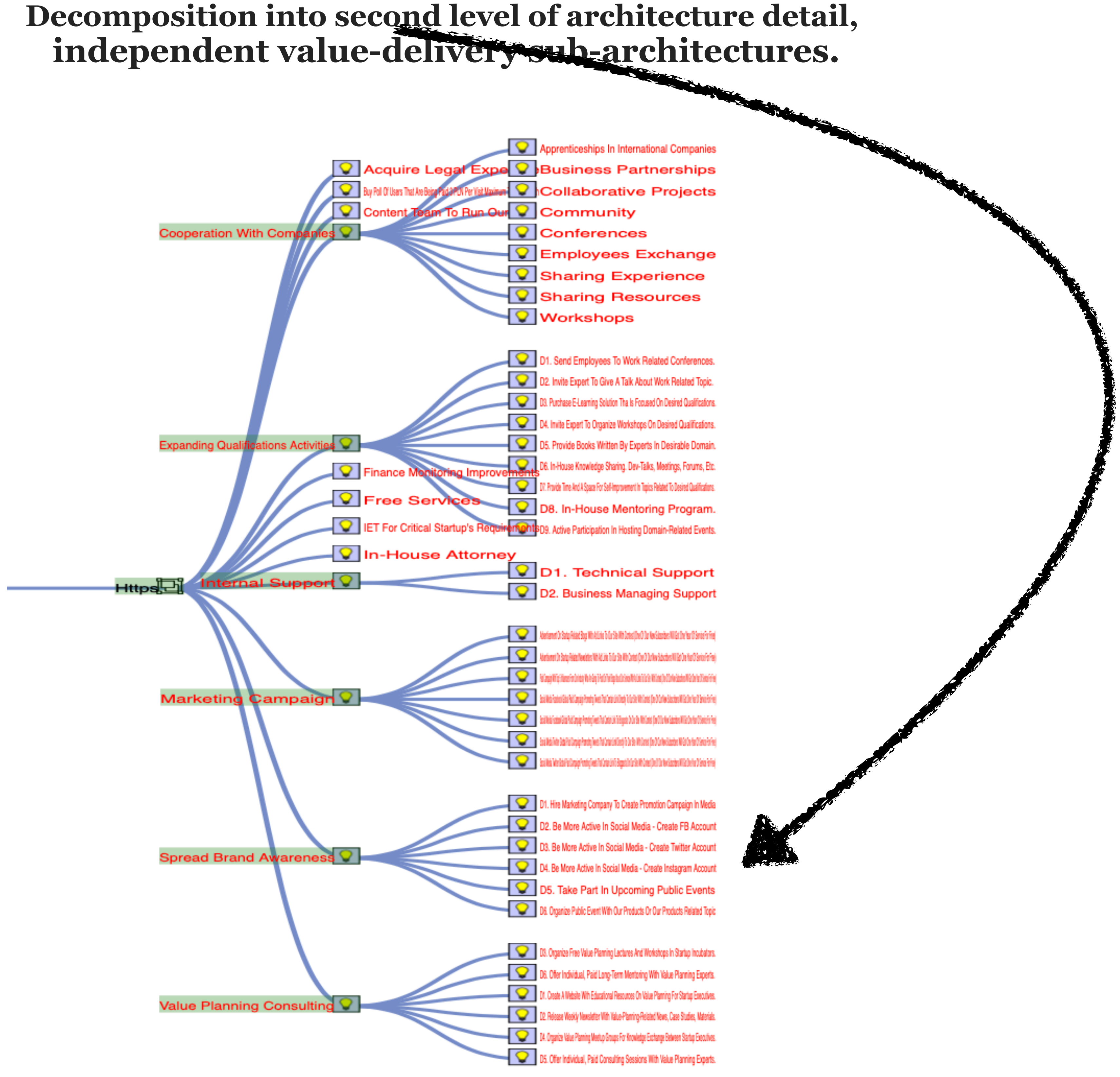

Https
Cooperation With Companies
Acquire Legal Expe
Buy Poll Of Users That Are Being Paid 3 PLN Per Visit Maximum
Content Team To Run Our
Apprenticeships In International Companies
Business Partnerships
Collaborative Projects
Community
Conferences
Employees Exchange
Sharing Experience
Sharing Resources
Workshops
Expanding Qualifications Activities
D1. Send Employees To Work Related Conferences.
D2. Invite Expert To Give A Talk About Work Related Topic.
D3. Purchase E-Learning Solution Tha Is Focused On Desired Qualifications.
D4. Invite Expert To Organize Workshops On Desired Qualifications.
D5. Provide Books Written By Experts In Desirable Domain.
D6. In-House Knowledge Sharing. Dev-Talks, Meetings, Forums, Etc.
D7. Provide Time And A Space For Self-Improvement In Topics Related To Desired Qualifications.
D8. In-House Mentoring Program.
D9. Active Participation In Hosting Domain-Related Events.
Finance Monitoring Improvements
Free Services
IET For Critical Startup's Requirements
In-House Attorney
Internal Support
D1. Technical Support
D2. Business Managing Support
Marketing Campaign
Spread Brand Awareness
D1. Hire Marketing Company To Create Promotion Campaign In Media
D2. Be More Active In Social Media - Create FB Account
D3. Be More Active In Social Media - Create Twitter Account
D4. Be More Active In Social Media - Create Instagram Account
D5. Take Part In Upcoming Public Events
D6. Organize Public Event With Our Products Or Our Products Related Topic
Value Planning Consulting
D3. Organize Free Value Planning Lectures And Workshops In Startup Incubators.
D6. Offer Individual, Paid Long-Term Mentoring With Value Planning Experts.
D1. Create A Website With Educational Resources On Value Planning For Startup Executives.
D2. Release Weekly Newsletter With Value-Planning-Related News, Case Studies, Materials.
D4. Organize Value Planning Meetup Groups For Knowledge Exchange Between Startup Executives.
D5. Offer Individual, Paid Consulting Sessions With Value Planning Experts.

Figure 4.1 D Source Polish Export Workshop 2017 Warsaw. There were 13 parallel teams (total about 60 people) working on the Architecture in parallel, in a single day, including decomposition and Impact Estimation.

The 3-person team suggested the 'Expanding Qualifications Activities' as their *main* architecture.
They were then asked 'to decompose into about 10 sub-architectures' (D1 to D9)
And later, in the afternoon, to evaluate these sub-architectures, on an Impact Estimation Table (see Figure 4.4B below)

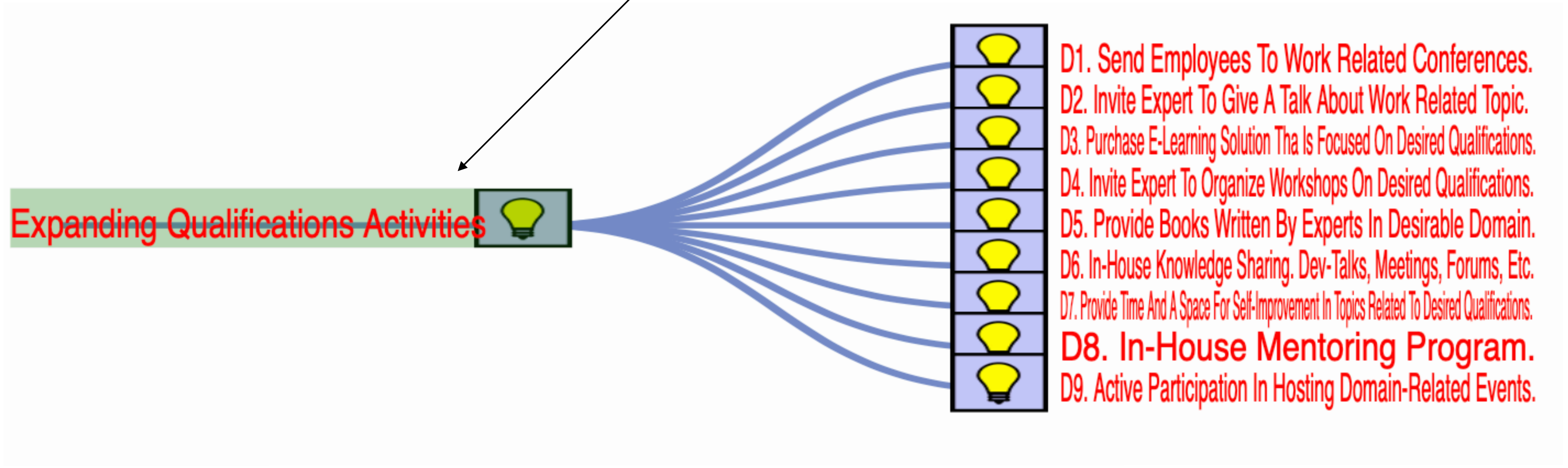
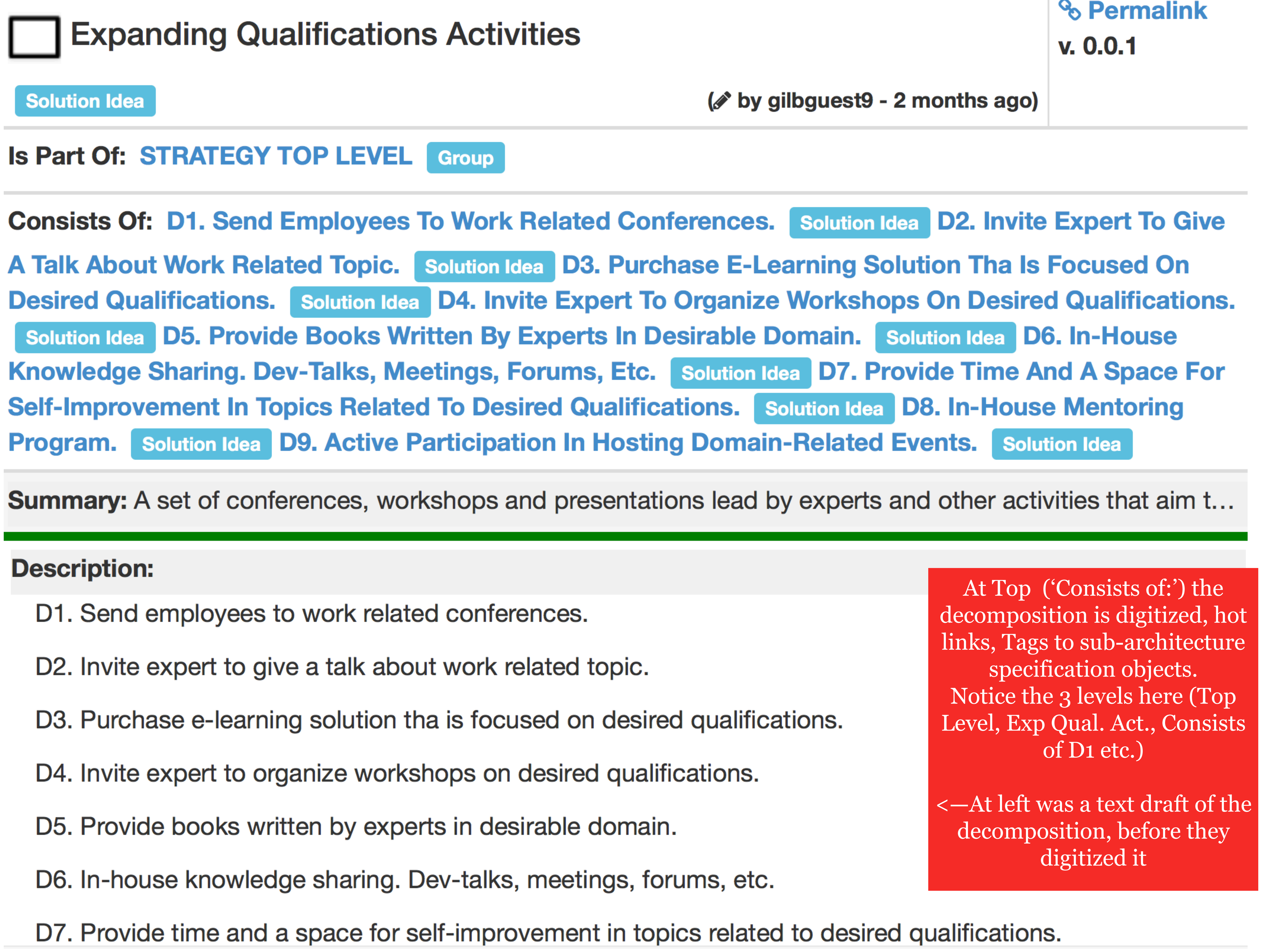

Expanding Qualifications Activities
D1. Send Employees To Work Related Conferences.
D2. Invite Expert To Give A Talk About Work Related Topic.
D3. Purchase E-Learning Solution Tha Is Focused On Desired Qualifications.
D4. Invite Expert To Organize Workshops On Desired Qualifications.
D5. Provide Books Written By Experts In Desirable Domain.
D6. In-House Knowledge Sharing. Dev-Talks, Meetings, Forums, Etc.
D7. Provide Time And A Space For Self-Improvement In Topics Related To Desired Qualifications.
D8. In-House Mentoring Program.
D9. Active Participation In Hosting Domain-Related Events.

Figure 4.2 Architecture 2nd Level decomposition Polish Export 2017



☐ Expanding Qualifications Activities

Permalink
v. 0.0.1

Solution Idea

(by gilbguest9 - 2 months ago)

Is Part Of: STRATEGY TOP LEVEL Group

Consists Of: D1. Send Employees To Work Related Conferences. Solution Idea D2. Invite Expert To Give A Talk About Work Related Topic. Solution Idea D3. Purchase E-Learning Solution Tha Is Focused On Desired Qualifications. Solution Idea D4. Invite Expert To Organize Workshops On Desired Qualifications. Solution Idea D5. Provide Books Written By Experts In Desirable Domain. Solution Idea D6. In-House Knowledge Sharing. Dev-Talks, Meetings, Forums, Etc. Solution Idea D7. Provide Time And A Space For Self-Improvement In Topics Related To Desired Qualifications. Solution Idea D8. In-House Mentoring Program. Solution Idea D9. Active Participation In Hosting Domain-Related Events. Solution Idea

Summary: A set of conferences, workshops and presentations lead by experts and other activities that aim t…

Description:

D1. Send employees to work related conferences.

D2. Invite expert to give a talk about work related topic.

D3. Purchase e-learning solution tha is focused on desired qualifications.

D4. Invite expert to organize workshops on desired qualifications.

D5. Provide books written by experts in desirable domain.

D6. In-house knowledge sharing. Dev-talks, meetings, forums, etc.

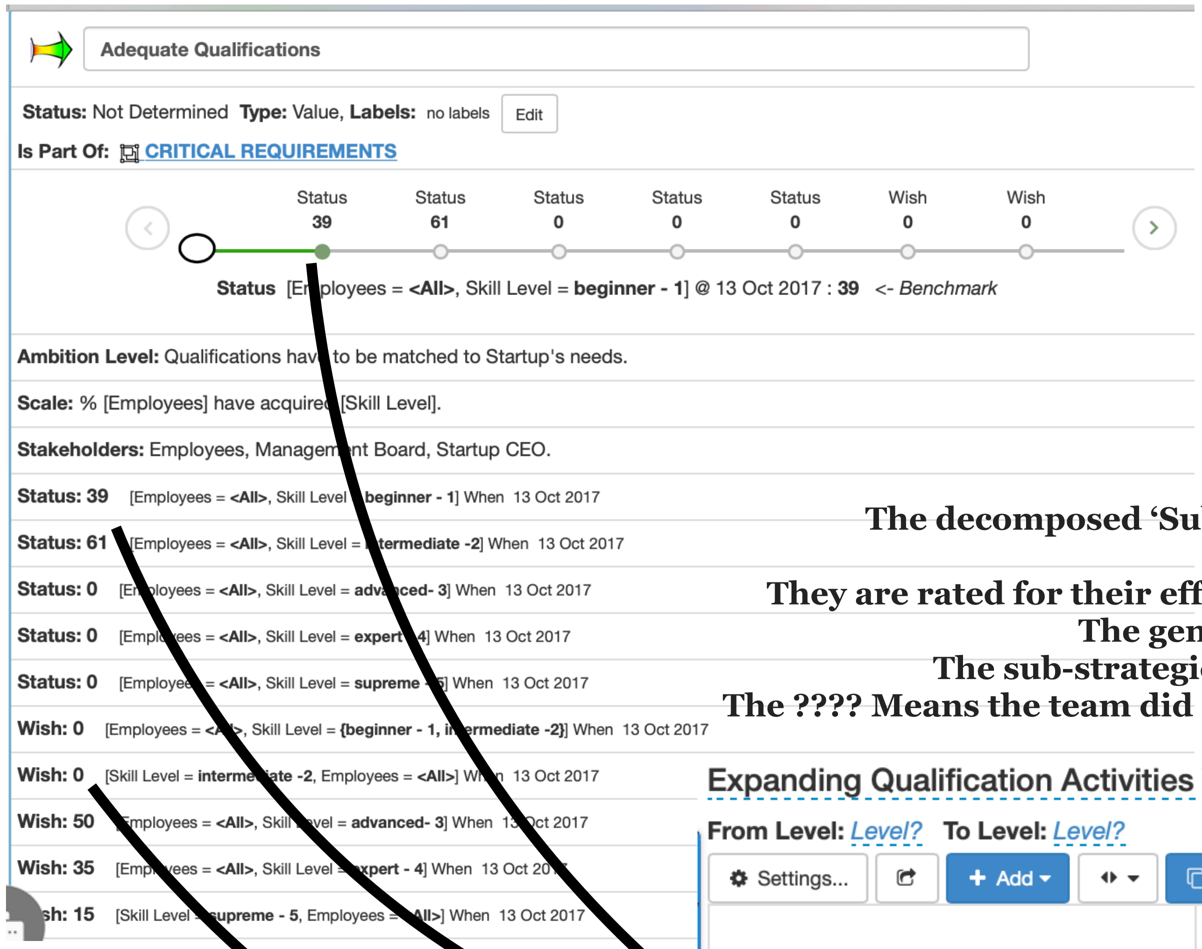
D7. Provide time and a space for self-improvement in topics related to desired qualifications.

At Top ('Consists of:') the decomposition is digitized, hot links, Tags to sub-architecture specification objects.
Notice the 3 levels here (Top Level, Exp Qual. Act., Consists of D1 etc.)

<—At left was a text draft of the decomposition, before they digitized it

Figure: 4.3 Polish Export Example, December 2017

The Adequate Qualifications Value Requirement.
This one **requirement is decomposed** using [Scale Parameter] Conditions.
For example Skill Level =expert, and Skill Level = supreme



Adequate Qualifications

Status: Not Determined **Type:** Value, **Labels:** no labels Edit

Is Part Of: CRITICAL REQUIREMENTS

Status	Status	Status	Status	Status	Wish	Wish
39	61	0	0	0	0	0

Status [Employees = **<All>**, Skill Level = **beginner - 1**] @ 13 Oct 2017 : **39** <- *Benchmark*

Ambition Level: Qualifications have to be matched to Startup's needs.

Scale: % [Employees] have acquired [Skill Level].

Stakeholders: Employees, Management Board, Startup CEO.

Status: 39 [Employees = **<All>**, Skill Level = **beginner - 1**] When 13 Oct 2017

Status: 61 [Employees = **<All>**, Skill Level = **intermediate -2**] When 13 Oct 2017

Status: 0 [Employees = **<All>**, Skill Level = **advanced- 3**] When 13 Oct 2017

Status: 0 [Employees = **<All>**, Skill Level = **expert - 4**] When 13 Oct 2017

Status: 0 [Employees = **<All>**, Skill Level = **supreme - 5**] When 13 Oct 2017

Wish: 0 [Employees = **<All>**, Skill Level = **{beginner - 1, intermediate -2}**] When 13 Oct 2017

Wish: 0 [Skill Level = **intermediate -2**, Employees = **<All>**] When 13 Oct 2017

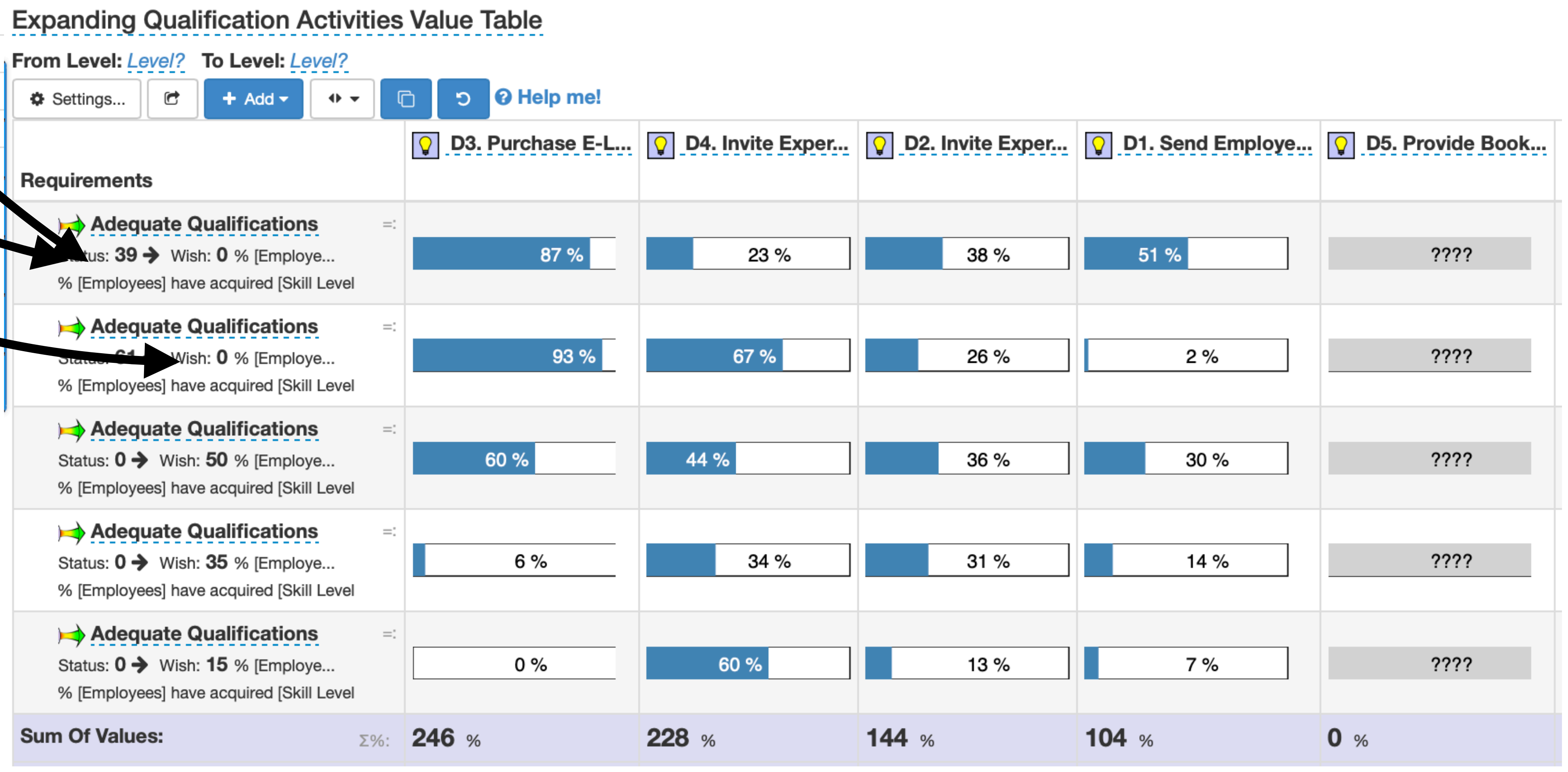
Wish: 50 [Employees = **<All>**, Skill Level = **advanced- 3**] When 13 Oct 2017

Wish: 35 [Employees = **<All>**, Skill Level = **expert - 4**] When 13 Oct 2017

Wish: 15 [Skill Level = **supreme - 5**, Employees = **<All>**] When 13 Oct 2017

Figure: 4.4 A. Polish Export Example 2017

The decomposed 'Sub-architectures' (D1 to D9) are listed at the top of an Impact Estimation Table (IET, see Part 12 here for detail on this method).
They are rated for their effect on a set of Sub Values (the Value 'Adequate Qualifications' is also *decomposed*).
The general sum of impacts on all Requirements are in 'Sum Of Values'.
The sub-strategies are *sorted* left to right for total impact. (Dynamic digital prioritization)
The ???? Means the team did not yet estimate the impact of D5. A 'Known Unknown'. Maybe the 'best': Who knows?



Expanding Qualification Activities Value Table

From Level: *Level?* **To Level:** *Level?*

Requirements	D3. Purchase E-L...	D4. Invite Exper...	D2. Invite Exper...	D1. Send Employe...	D5. Provide Book...
Adequate Qualifications Status: **39** → Wish: **0** % [Employe... % [Employees] have acquired [Skill Level	87 %	23 %	38 %	51 %	????
Adequate Qualifications Status: **61** → Wish: **0** % [Employe... % [Employees] have acquired [Skill Level	93 %	67 %	26 %	2 %	????
Adequate Qualifications Status: **0** → Wish: **50** % [Employe... % [Employees] have acquired [Skill Level	60 %	44 %	36 %	30 %	????
Adequate Qualifications Status: **0** → Wish: **35** % [Employe... % [Employees] have acquired [Skill Level	6 %	34 %	31 %	14 %	????
Adequate Qualifications Status: **0** → Wish: **15** % [Employe... % [Employees] have acquired [Skill Level	0 %	60 %	13 %	7 %	????
Sum Of Values: Σ%:	**246** %	**228** %	**144** %	**104** %	**0** %

Figure: 4.4 B. Polish Export Example 2017

The Sub-architecture Table (Figure 4.4 B) is here 'converted into a Bar Chart' for simplified presentation. Any spreadsheet can do this.
The Sub-architectures are not yet sorted by *effect* level (as they are in the Table).
The I figure at the bar top is the range of ± uncertainty, stated together with the estimate.
We can see, visually, and digitally, the best-case and worst-case levels are estimated.
The costs (right-side bar for each sub-arch) are not yet estimated. They will be used to evaluate the sub-arch efficiency.
In other words it is *too early* to declare a 'most efficient' sub-architecture. And prioritize it for delivery early.
The right-hand 5 sub-strategies have not yet been estimated at all. ????, Known Unknowns. More planning necessary.

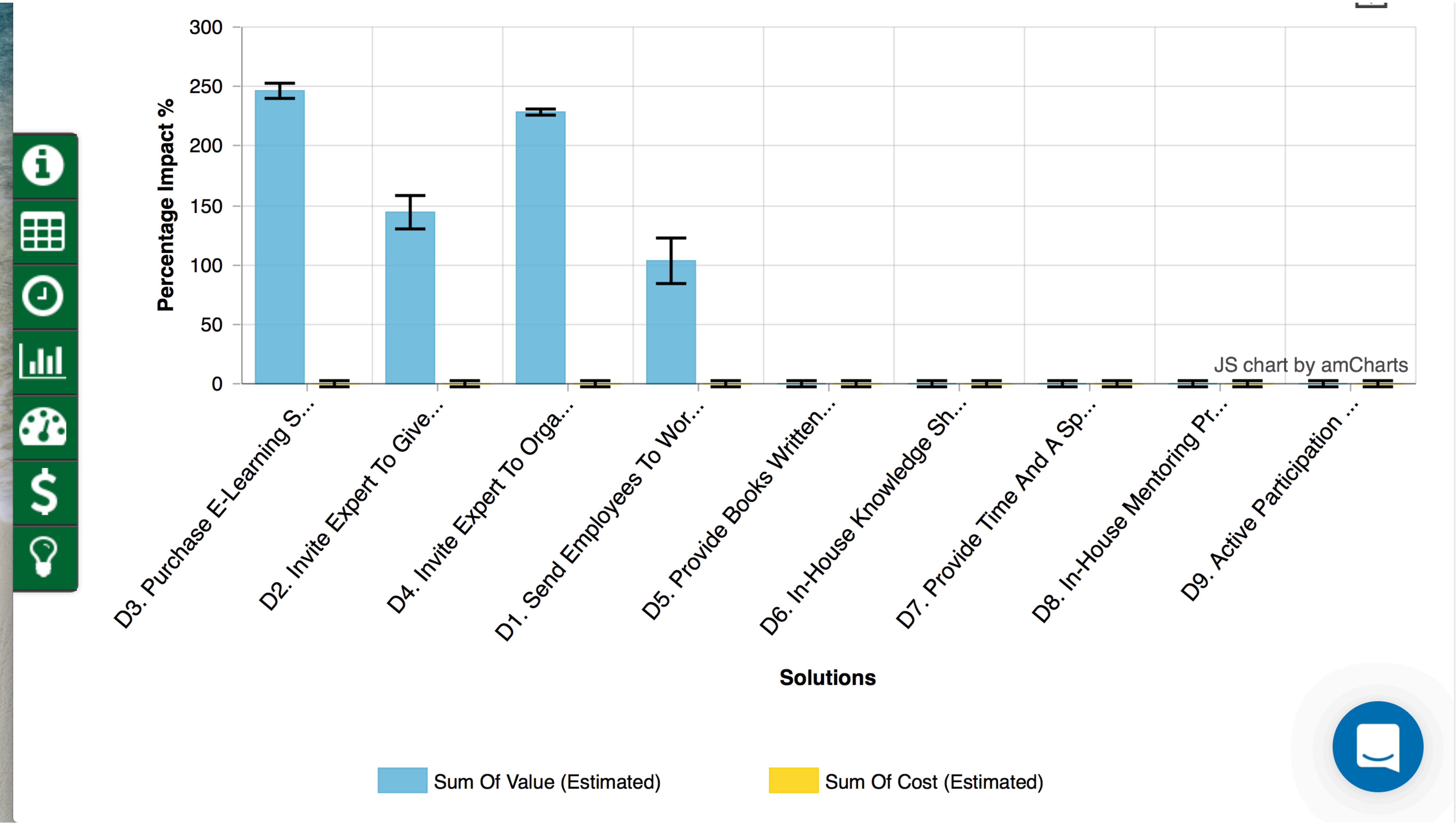
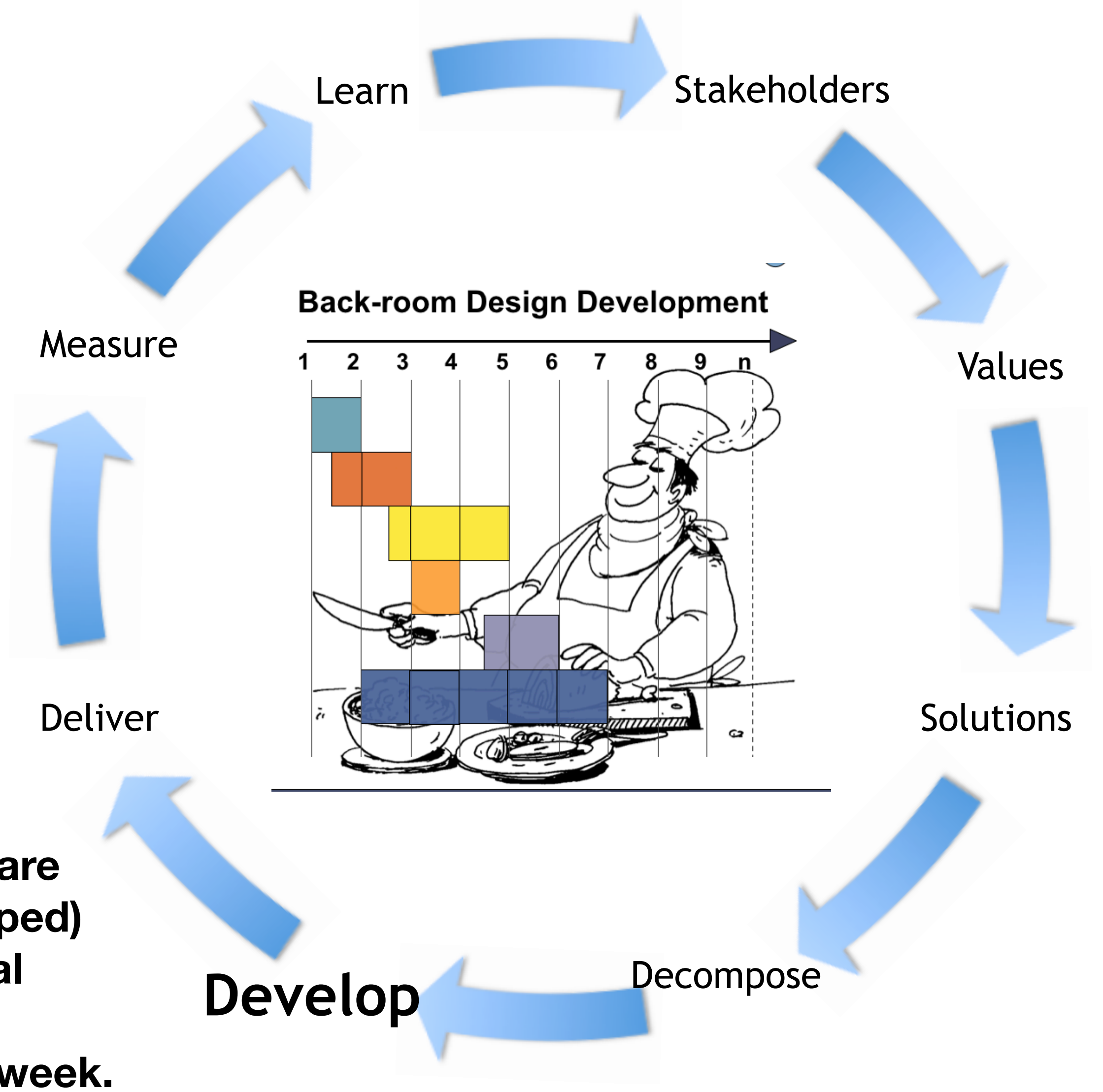


Figure: 4.5. Polish Export Example 2017

After 'decomposition, and estimation, we have digital information, that we can use, to prioritize sub-architectures, for development and delivery. Not all sub-architectures require ground-up development. They may only need activation, or purchase, or permissioning. In any case 'Develop' covers whatever must be done to allow 'Deliver', which is integration in a live system, so that we can get value delivered, and get feedback on side-effects, costs and stakeholder reactions ('Measure'), Then we can consequently 'Learn' how to *do things better*. It is not least the *architect*, who must be prepared to begin this Measure-Learning loop. Architects need to make adjustments to the architecture, in order to deliver sufficient effect, at acceptable costs. (Dynamic Design-To-Requirements. [P3.3 Quinnan]



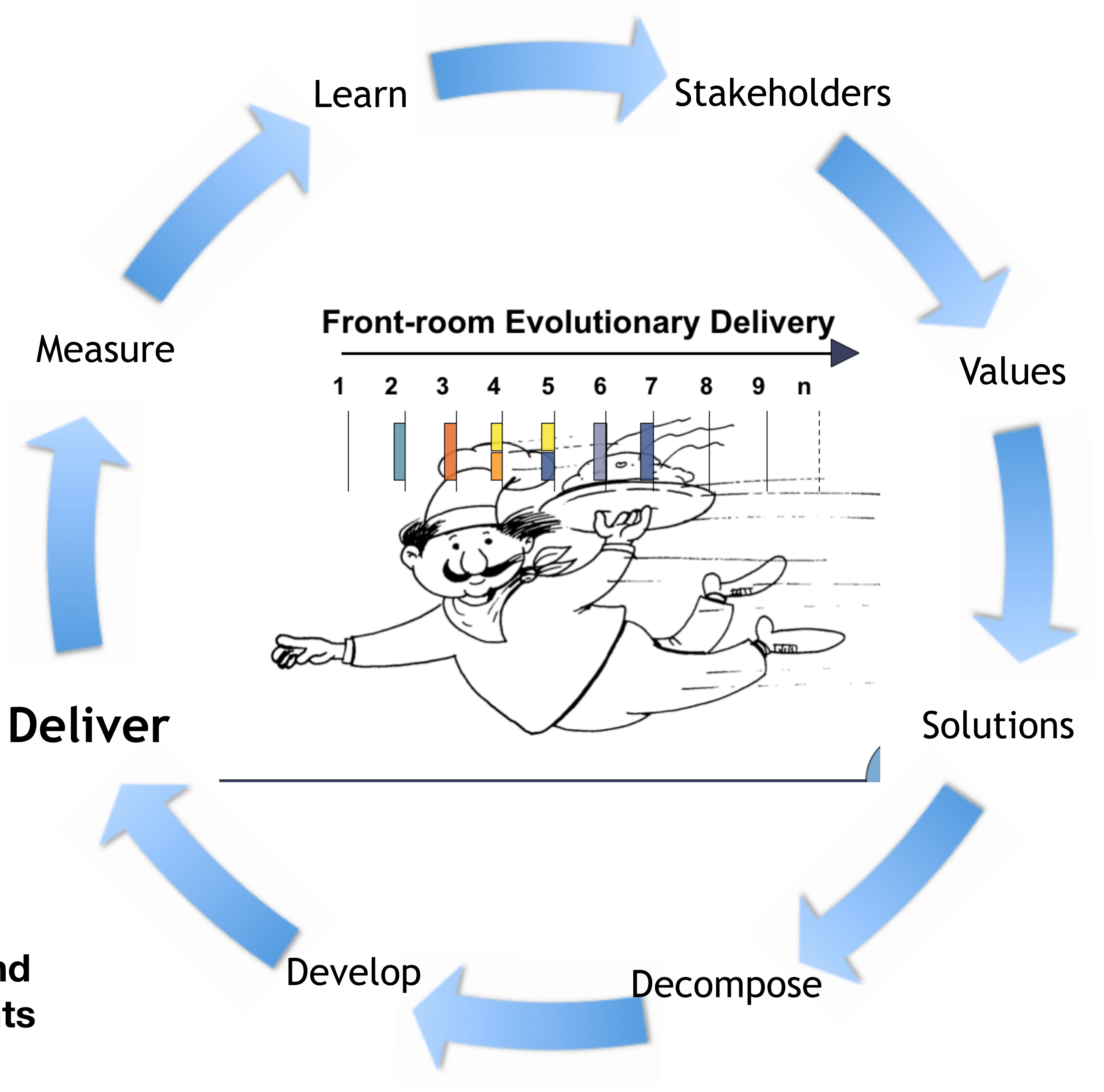
The sub-solutions are made ready (developed) for delivery to real stakeholders, next week and every week. Or in about 2% of budget/ deadline increments

X

Figure: 4.5 The Gilb Evo 'Value Delivery' Cycle

Sub-architecture 'Delivery' means inserting some aspect of the sub-architecture, not necessarily *all of it everywhere*, into a real system (maybe under tightly-controlled reversible conditions, if something goes wrong.

The main point is to try to deliver measurable value.
The secondary point is to be able to learn what is real, in main 'value', in delivery problems, in side-effects, in costs, and in stakeholder reactions.
And to quickly react, at architecture level, to make things better.



The sub-solutions are *delivered* to real stakeholders, in order to experiment, to test, to pilot, to get reactions, NUMERICALLY and to allow for potential corrections in design, in implementation process, and in lower-priority requirements

Figure: 4.6. The Gilb Evo 'Value Delivery' Cycle

Engineering (SEA Systems Enterprise Architecture is A. E.) is a *quantitative* set of tools, there are non-quantitative feedbacks possible at every cycle, for example complaints, delays, media reactions, insider competitor bickering, competitor bad-mouthing, political reactions, which are very important for the architect to take on board, and deal with. They are not directly *our* values and costs, which we *want* to measure. And we can count (complaints) to a degree. But the architect is in charge of the whole big picture, and must be prepared to sense all manner of feedback, early: and consider architecture tuning, correction, or even 'pull the plug', reverse and find new architecture, before we scale-up a bad architecture. We are clearly dealing with the 'pilot studies' at *each* increment (but often quite *real* people and places) that guide us, as to whether we are ready to scale up, to the 'country,' and the 'world'.
This is, I think, the Cartesian Scientific Method [See 4.0, 2 pages].
Real 'Computer *Science*' and 'Software *Engineering*'.
A tough, but necessary, discipline to succeed, and to not fail 95% of the time [See Part 0.0 above]

The sub-solutions are
<u>measured as to effect</u>
on
all the
top
stakeholder
critical
objectives,
and
on their critical cost
increments,
with a view to improving
<u>prediction</u> of
final cumulative costs

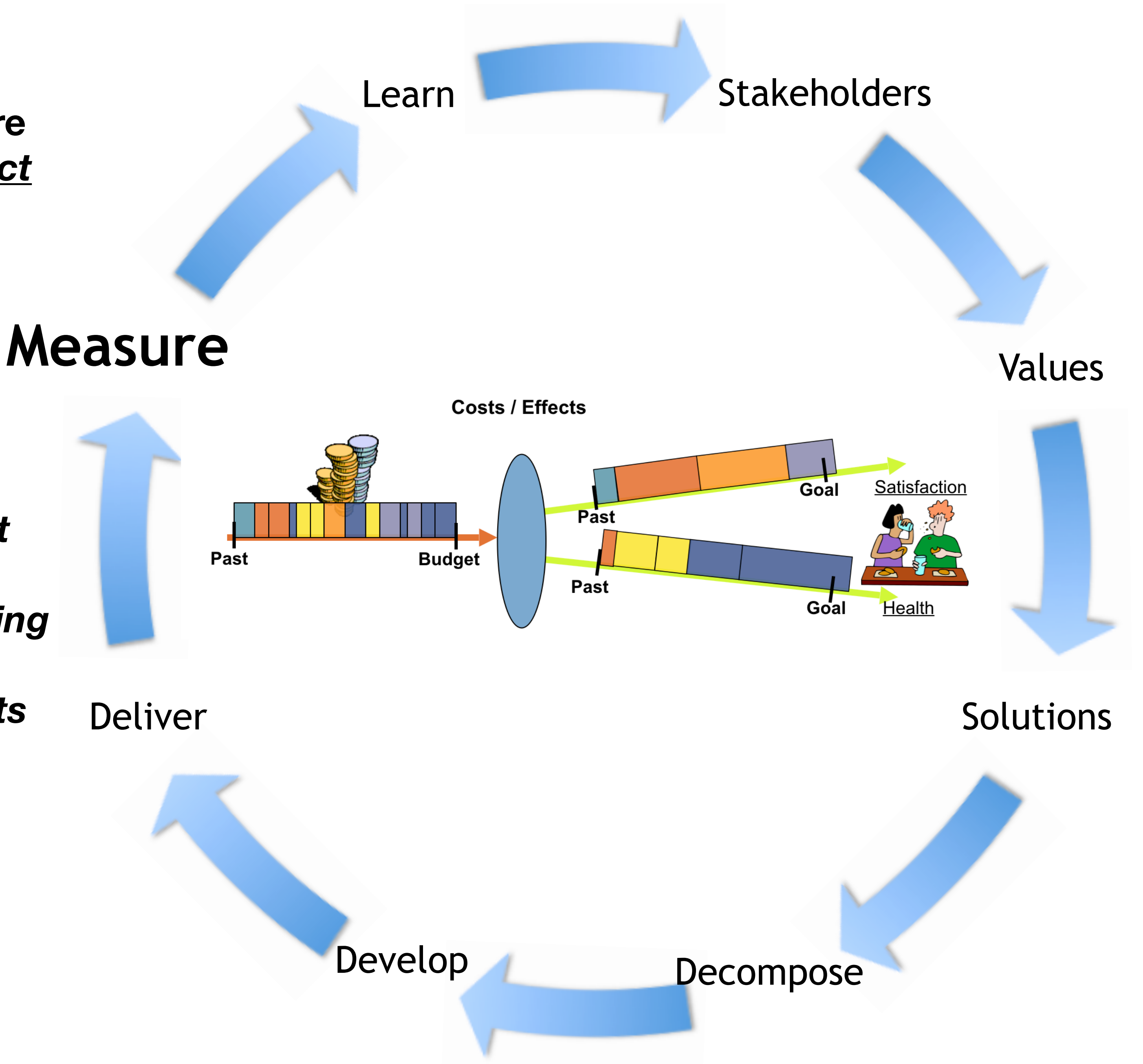


Figure: 4.7 The Gilb Evo 'Value Delivery' Cycle. Courtesy Kai Gilb.

We analyze the degree of deviation of values and costs, from our estimates, and our ± uncertainty or Landing Zone ranges. We look at other stakeholder reactions, expected and unexpected.

From this, the architect needs to possibly modify, or replace the architecture. Make no mistake: Systems Enterprise Architecture means (1) you have to look at all possibly useful-an- relevant deviation signals, from all possibly-interesting sources. That might mean you need to (2) specify as part of your architecture, the data-collection and measurement devices that must accompany your other architecture, into implementation and operation.

These are necessarily an intimate and critical component of the architecture. The architect must be *trained* (to architect architecture-sensors), and be *equipped* (like checklists, rules, sensing tools) to do this, as a part of their job.

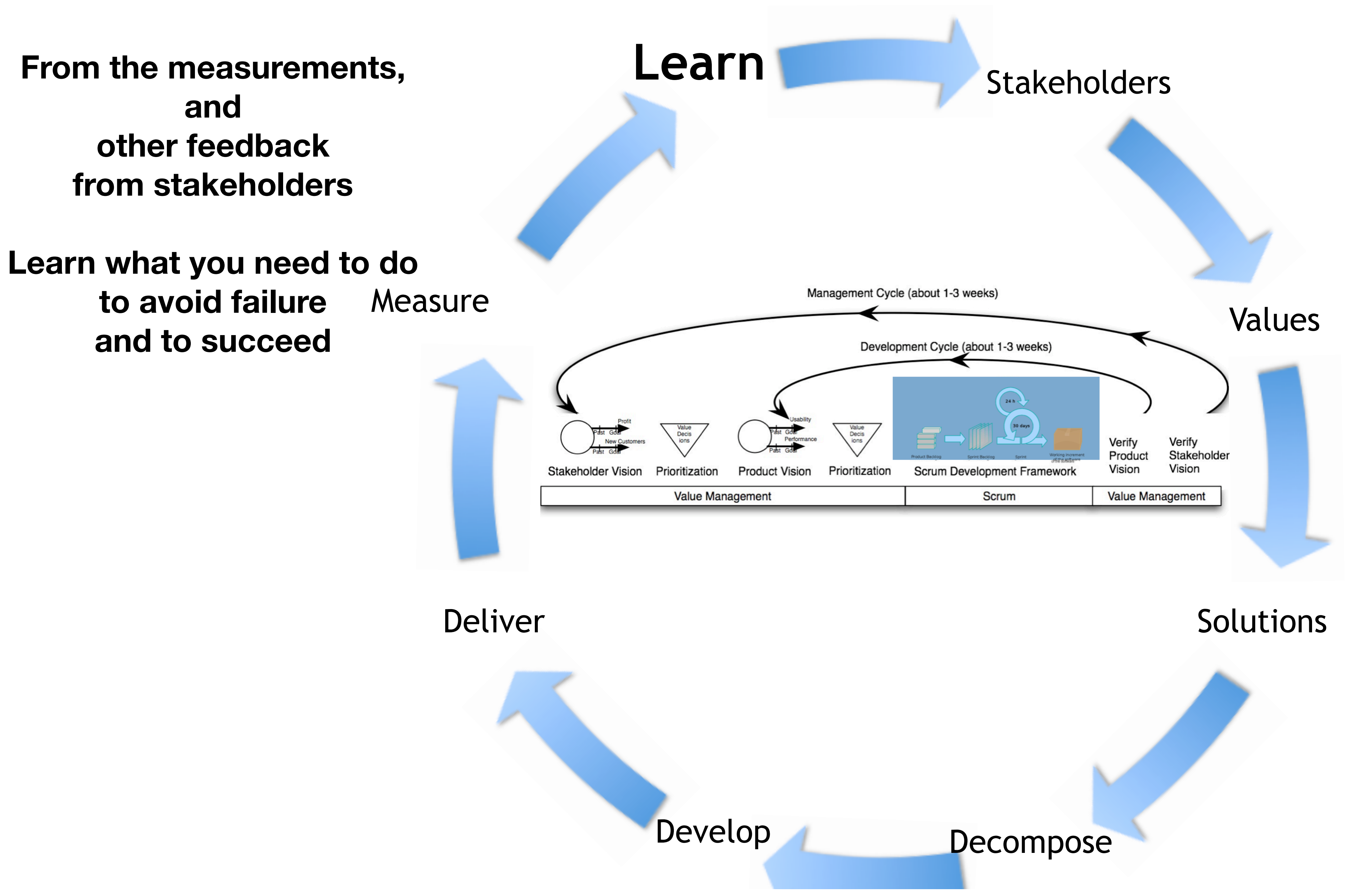

From the measurements, and other feedback from stakeholders
Learn what you need to do to avoid failure and to succeed
Learn
Stakeholders
Values
Solutions
Decompose
Develop
Deliver
Measure
Management Cycle (about 1-3 weeks)
Development Cycle (about 1-3 weeks)
Stakeholder Vision
Prioritization
Product Vision
Prioritization
Scrum Development Framework
Verify Product Vision
Verify Stakeholder Vision
Value Management
Scrum
Value Management

Figure: 4.8. The Gilb Evo 'Value Delivery' Cycle . Original diagrams Kai Gilb.

4.3 Decomposition: part of Success

Why is this Evo/Cleanroom method so good?

A common sense explanation

I would like to explain, in a focussed way, why this 'continuous agile architecture method' (aka Evo, Cleanroom) is so guaranteed to lead to success, or at least avoid failure.

1. We **decompose** implementation into small '2%' real delivery increments. The biggest failure we can have, is 2%. *When I gamble in Las Vegas, I bet $10 on Red. Walk away no matter whether I win or lose. Got it?*

2. We *measure* the ***value*** results (not the 'code or stories' delivered) and the costs, at *each* step. Not at the 'end', at each step.

3. The architect is **in the loop**, and has the **power** to correct the **architecture**, to ensure value and costs. All real values-and-costs power, is **in the design**!

4. We try out the improved architecture ***immediately***, to *make sure it works*.

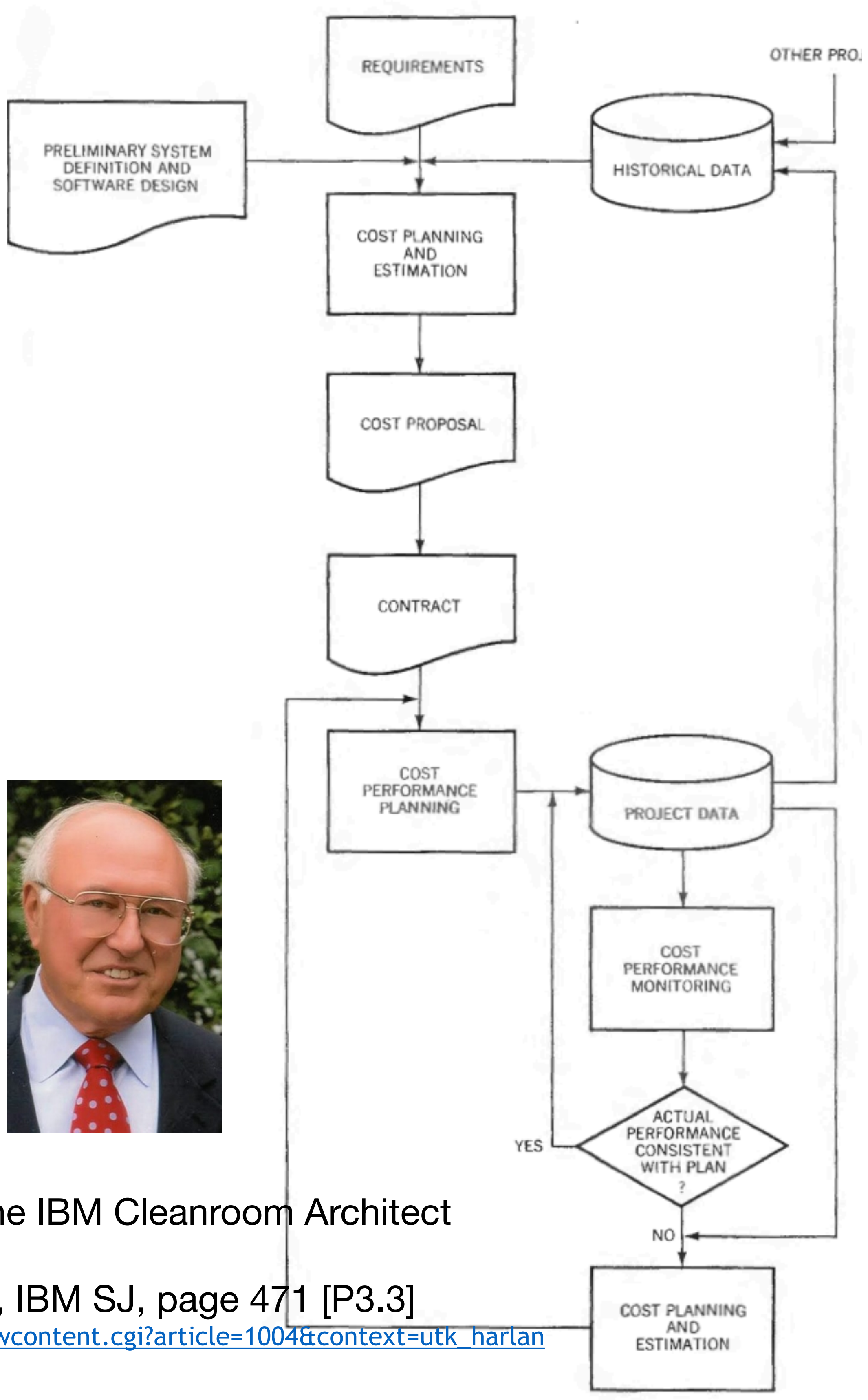
5. We do **not** persist, or scale up, with things that do *not* work! What was Einsteins's definition of a fool?

6. We **cumulate value** success incrementally. We *ratchet* it in.

7. We (next Part 5 Prioritization numerically **prioritize** the best value/cost and risks, so that even with disappointments, things *probably* deliver good values, and low costs.

8. If a *series* of small increments all fail, we clearly do not know what we are doing. So we **stop the project** until a new competent architect, can specify sub-architectures that *can* deliver values at good costs. You *cannot fail big* (see 0.0) if you stop, when bad results persist. Albert again.

Figure 1 Cost management process


REQUIREMENTS
OTHER PRO.
PRELIMINARY SYSTEM DEFINITION AND SOFTWARE DESIGN
HISTORICAL DATA
COST PLANNING AND ESTIMATION
COST PROPOSAL
CONTRACT
COST PERFORMANCE PLANNING
PROJECT DATA
COST PERFORMANCE MONITORING
YES
ACTUAL PERFORMANCE CONSISTENT WITH PLAN ?
NO
COST PLANNING AND ESTIMATION

Robert Quinnan, The IBM Cleanroom Architect

Source: Quinnan, IBM SJ, page 471 [P3.3]
http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan

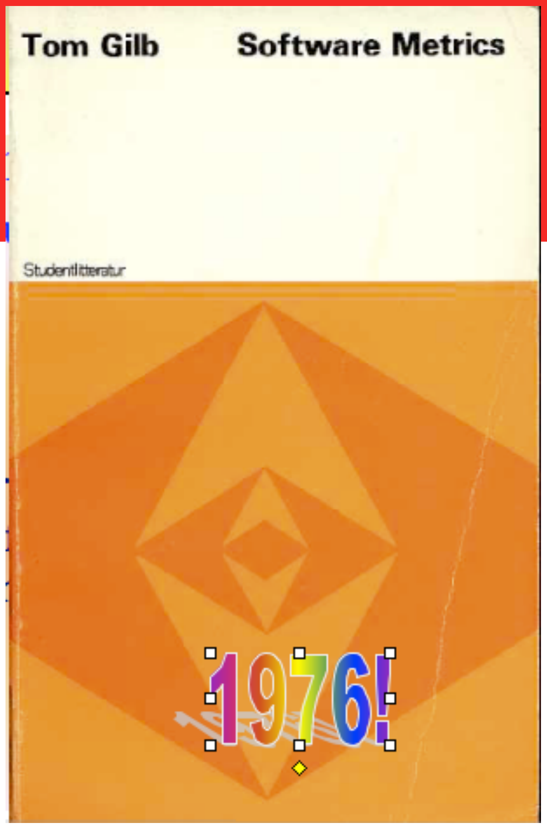
4.3 Decomposition: part of Success, the other parts are…

Let me summarize the previous page with the key points related to the Systems Enterprise Architecture. What are the key things we do to manage success and avoid EA failure? What are the methods that your EA methods are **not** doing (you need the *whole* set below!)?

1. Decomposition into **deliverables, that deliver value**
2. Numeric **Prioritization: Values, Costs, Risks** (IET) [Part 5]
3. **Numeric Feedback every increment**: measure Value
4. **Architect in the loop**, with power to immediately change bad architecture.
5. **Cumulate numeric values** until Goals reached
6. **Power to stop** the whole effort, when your incremental failure frequency is too high.
7. Scale up, **only when smaller-scale success is measurably proven**, is l'ocked in', ratcheted in.
8. If an EA architect 'cannot even design one simple 2% step to deliver measurable stakeholder value', they are **not a competent EA architect**. Use this as a test of architecture value knowledge.

'A complex system
will be most successful,
if it is implemented in small
steps,
and if *each* step has
a clear measure of successful achievement,
as well as a 'retreat' possibility
to a previous successful step,
upon failure.'

(Gilb, Software Metrics book, 1976 p. 214)
(are you listening yet?).
or do you need another 44 years to 'get it?'



USA edition 1977.

This, above, is agile,
' as it *should* be'.

[VA]

Why have you not heard about 'Evo' [Part 5 here] and Cleanroom, which are decades old, and well-documented in public?
That is a very interesting discussion. Ask your professors. 🧑‍🏫 Anyway, *you* are HERE now, and this is the 'most powerful single thing' I can teach you.

Each Evolutionary Cycle

consumes a budget of Development Resources.

We need to keep our eyes on

something like 14 critical top-level value-and-resource requirements *simultaneously*.

So we need *tools, tables and numbers* to help us to keep track of it all,

both for each architect, and as dispersed teams

This diagram (Kai Gilb) is intended to hep you visualize
the 'cycles of incremental value improvements',
concurrent with 'consumption of limited, budgeted resources'.
In real life, we keep track of this *digitally*. Even if only in a spreadsheet, as in Figure 5.1.

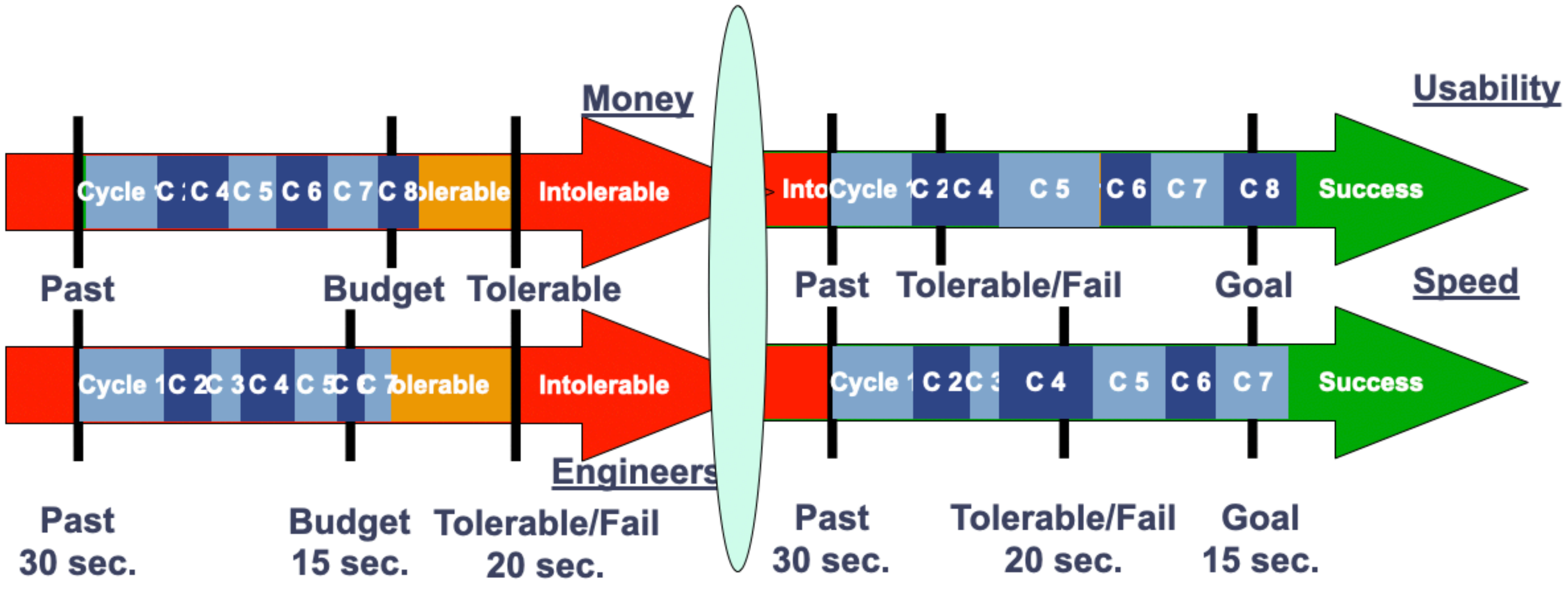


Figure: 4.9 The Gilb Evo 'Value Delivery' Cycle viewed as a series of increments delivering values, and using resources.

Deciding what to do now.

Dynamic Architecture Prioritization

I am going to be brief, about prioritization, in this book, but if you want more depth on 'dynamic prioritization', see these References. Free downloads.

P14. Managing Priorities, A Key to Systematic Decision-making. With Mark Maier, 2005 (paper, 19 pages) http://www.gilb.com/DL60

B2: Value Planning, Chapter on **Prioritization, 60 pages**: https://www.dropbox.com/sh/34llx1a7ckyagxl/AAA0pDzSxN5Wm0P9lOKR0Mpca?dl=0

See these 'Prioritization' references in 'References', below.

Most people , books and processes have a 'static' concept of prioritization. It is usually based on some kind of fixed, subjective, anonymous, 'weights' to indicate the 'priority', with no written justification to indicate why the weights were set, let alone who is responsible.

I find such prioritization methods, anti-agile, weak, potentially dangerous, childish, and simplistic for large-complex systems. Maybe weights are OK in a simple setting such as personal decisions. But *complex evolving systems* need a *better* method.

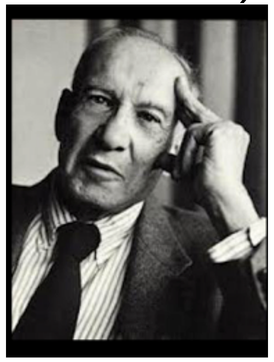
In one sense, there *is* a better method; we all know quite well. It is built in to all living things, and especially humans. We humans determine priorities, not on 'fixed weights', but on our multiple values, and our consumption of limited of resources. We do so in changing circumstances, in unpredictable, situations, using quite 'multidimensional value' thinking, combined with quite multi-dimensional resource-thinking. Biology is pretty good at this. It is obviously in our DNA, and we can get better at it with experience. So, that is my *Dynamic Architecture Prioritization*: we just need to make it *operational for the architecture environment*. That is easy, because the SEA Language has tools built in to do it, formally, transparently, rigorously, for complex evolving systems.

5. Architecture Prioritization

Drucker outlines an ‘enterprise’ framework: the same one I am suggesting in this book.

“Work implies not only that somebody is supposed to do the job, but also accountability, a deadline and, finally, the measurement of results —that is, feedback from results on the work and on the planning process itself.”

Drucker wrote in ‘Management: Tasks, Responsibilities, Practices’. Peter Drucker,: Business Author (1909 Vienna - 2005 Claremont, California).



It is amazing how much of what I am suggesting in the SEA framework, is condensed into this single sentence.
Drucker is one of the most respected management writers. I met him once, and got him to sign his book.
He was delighted (in San Diego) to talk to a ‘fellow European’.

Let me rephrase him, so you cannot miss <u>my</u> point:

<u>Doing</u> work is <u>not</u> the central point of work.
Somebody has to take <u>responsibility</u> for the results.
And the results are MEASURABLE!
And there is FEEDBACK from the results.
And the FEEDBACK is to be used, to impact the planning process itself!

Wow, this man has his head screwed on straight.
Is he ‘agile’? With quantified result-feedback to help planning?
Yes. Real agile management, as it should be.

The assumptions

1. **DECISION OPTIMIZATION:** You want the *best architecture choices*, even if you have to work a little harder to get them.
2. **TRANSPARENCY:** You want, or need, your *prioritization decisions* to be *transparent*, because of enterprise governance, and law.
3. **VALUE SPEED**: You need to speed up *delivery of values*.
4. **RESOURCE MINIMIZATION:** You need to 'be seen' to *minimize resources*, which are needed to 'get to your value levels', for your economic and ethical reasons.
5. **LEARN**: Complex technology or social environments demand, that you can *constantly learn* how your prioritization decisions, actually worked out, and *why*. So the prioritization methods themselves need *continuous updating*.
6. **RESPONSIBILITY:** You need clear ways to assign responsibility to individuals, and groups, for their 'priority decisions'.
7. **POLICY LOYALTY:** You would like to be sure that enterprise prioritization policy is *actually followed*, and is not 'someones intuition'.
8. **DIGITIZATION**: you would like to *integrate* prioritization *digitally*, with all *other* forms of planning.
9. **AUTOMATED DECISIONS:** you would like to automate the 'selection of prioritized architecture specs'; and you want to 'enable a consistent stream of priority evaluations': where you can use a more-complicated set of factors, than normal human decision-making. For example, avoiding over-simplistic 'focus on *only one value and one cos*t'.
10. **AGILITY:** Rapid response to changed situations (Agility). You are interested 'rapid-response decision-making to changed circumstances', for public-health or economic reasons, for example.

5.1 Architecture

Agile prioritization

"I can't change the direction of the wind, but I can adjust my sails to always reach my destination."

Jimmy Dean (August 10, 1928 – June 13, 2010) was an American country music singer, television host, actor, and businessman.

This poetic statement is a good analogy:

set clear long-term quantitative goals ("my destination"),

and in spite of competition, negative stakeholders, and regulation ("direction of the wind"),

I can use feedback and course correction ("adjust my sails").

This sounds like 'Dynamic Design to Cost' I am discussing here ([P3], Cleanroom, Evo [B1])

5.2 Architecture Prioritization

The Mechanics

You have already, in this book, seen most everything you ‘need to use’ in order ‘*to prioritize*’. Here is the dynamic agile prioritization framework.

1. **THE DECISION BASIS**
   a. **<u>Value requirements</u>: quantified, structured, rich.**
   b. **<u>Resource limits</u>, budgets, long term, short term, all types of resources.**
   c. **<u>Constraints</u>: knowing when a choice is invalid.**
2. **THE CHOICES**
   a. **Architecture, Sub-architecture.**
   b. **Selected stakeholders**
   c. **Selected Values, for selected Conditions.**
3. **THE DECISION POLICY ELEMENTS**
   a. **Objectivity: truth, or the truth about bad evidence**
   b. **Multiple-Factor Consideration**
   c. **Values, Resources, Risks**
4. **THE DECISION PROCESS**
   a. **Total automation**
   b. **Automatic suggestions. Human approval.**
   c. **Automatic suggestion *options*, human responsible choice**
   d. **Deviant /exception choices, based on 'documented justification’ and ‘responsibility’.**
   e. **Artificial Intelligence, learning about best-policies.**

5.2 Architecture Prioritization: 'Fact-Based Architecture', anyone?

"Facts are stubborn things;

and whatever may be our wishes, our inclinations, or the dictates of our passions,

they cannot alter the state of facts and evidence."

John Adams (1735-1826), 2nd President of US

5.3 Architecture Prioritization: Principles

From: Value Planning, Chapter 6, on **Prioritization,** 60 pages: https://www.dropbox.com/sh/34llx1a7ckyagxl/AAA0pDzSxN5Wm0P9lOKR0Mpca?dl=0
See this, for *detail* on these principles

PRIORITY PRINCIPLES

from Value Planning book, Chapter 6.

Principle 6.1 You might Get it all, if…

If you have *infinite* resources, you can have it all – 'choices' are not necessary

Principle 6.2 DYNAMIC PRIORITY:

'Static initial prioritization' is unrealistic – things change

Principle 6.3 PRIORITIES POLICY

You need a clear prioritization *policy*, and it can *change* and *vary*.

Principle 6.4 BANGS FOR BUCKS: THE EFFICIENT PRIORITY PRINCIPLE

A good, general, prioritization policy, will deal with the *critical few* objectives, the *most powerful* strategies, and their *numeric relationships*, of values-to-resources.

Principles 6.5 Data-Driven Decisions

An Impact Estimation Table gives you a systematic overview of the many related facts, about your potential priorities.

Principle 6.6 LONG-TERM, AND SHORT-TERM VALUE

An Impact Estimation Table will help you prioritize, in the short-term *and* long-term

Principle 6.7 Estimates are a good start, measurements are better.

An Impact Estimation Table can help keep track of 'real feedback' and of your 'incremental progress' towards Targets..

Principle 6.8 PRIORITY LOGIC.

Priority can be 'computed' by an Impact Estimation Table

Principle 6.9 NATURE'S PRIORITY

First priority is survival, to get to 'tolerable levels'. and avoid 'constraints'.

Principle 6.10 Dreams are not enough

You cannot commit to mere 'stakeholder wishes', they need to be implementable in *practice*.

5.3 Architecture

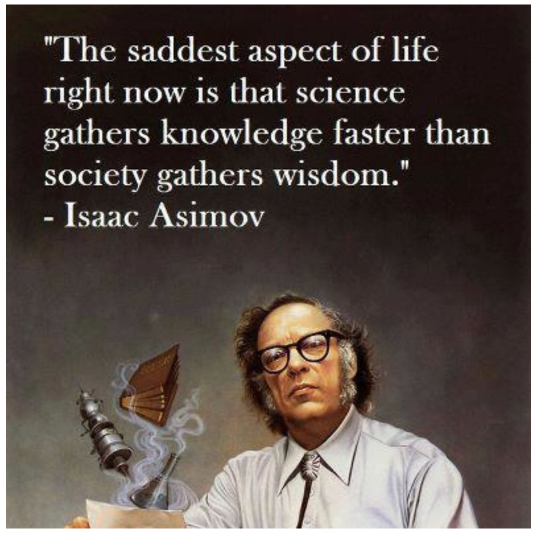
"I believe in evidence.

I believe in observation, measurement, and reasoning, confirmed by independent observers.

I'll believe anything, no matter how wild and ridiculous, if there is evidence for it.

The wilder and more ridiculous something is, however, the firmer and more solid the evidence will have to be."

Isaac Asimov (2 Jan 1920 - 6 Apr 1992).


"The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom."
- Isaac Asimov

5.4 Architecture Prioritization: Policies

Priority Policies

There is no *one* ‘prioritization policy’, or ‘rules for how we should prioritize’.

It possible to have *any useful number* of policies for prioritization; and to apply them at *any opportune time* in the architecture process.

You can prioritize specific **stakeholders**. You can prioritize specific **values**. You can prioritize specific **architectures**. You can prioritize finding architectures **which make use of abundantly remaining resources**, as you near the end of an incremental delivery sequence, which has depleted some resources, but not others.

This agile fact (use of remaining resources), is very interesting, because it means that you *might be forced to select*, **totally different architectures** than *initially* anticipated, late in the delivery process, as you approach your Target levels, because you ‘surprisingly' have depleted *some* resources, *and* have *more than enough* of other resources.

The corollary of this proposition is, that *premature selectio*n and *fixed decisions*, for some ‘otherwise attractive architectures’, might be disastrous for the **resources**, *towards the end* of an incremental value-delivery sequence.

This is the old ‘decision-making principle’, of delaying final decisions about anything, until the last possible moment, so that your have a maximum of facts available, to make a smart decision. That is why Presidents Procrastinate about using Nuclear Weapons.

A good general policy for prioritization is to try to get *maximum value for minimum resources*. Similarly, but trickier, is *maximizing* a ‘set of Values’ (like the top-10 critical value-objectives for a project), *in relation to* a ‘set of the top 5 resources’. Values/Costs. This is built in to the ValPlan tool, and can be easily tracked on any spreadsheet.

Even more useful, because we have so-little credible data about our architecture’s value attributes, and about architecture cost attributes, in high tech, is to take into account the *riskiness* of the data we use, to make a priority decision.

We use the ± Uncertainty, and the Evidence Credibility (0.0 to 1.0), which is part of the SEA Language (B1), and built into the ValPlan app, to take risks in consideration. [See Part 7].

5. Architecture Prioritization

Creating a highly profitable enterprise, by a man who did it.

I don't think that you can invent on behalf of customers unless you're willing to think long-term, because a lot of invention doesn't work.

If you're going to invent, it means you're going to experiment, and if you're going to experiment, you're going to fail, and if you're going to fail, you have to think long term.

Jeff Bezos, Amazon (1964-   )

GOOGLE'S SHOPPING LIST · WHY TO SELL NOW
Forbes
"OUR CULTURE IS FRIENDLY AND INTENSE, BUT IF PUSH COMES TO SHOVE WE'LL SETTLE FOR INTENSE."
AMERICA'S BEST LEADERS
THE SECRET TO JEFF BEZOS' CUSTOMER-DRIVEN IDEA MACHINE
PLUS: THE GOOD, THE BAD AND THE OVERPAID (THIS MEANS YOU, MEL KARMAZIN)

What is he saying about enterprise architecture?

It sounds to <u>me</u> like my 'value agile, Dynamic Design to Cost' idea [P3] again

Suck it and see, oops, sorry 'scientific experiment'.

5.5 Architecture Prioritization: Reality Beats Modelling

Suck it and See

There is a limit to usefulness of architecture models, even the exciting Impact Estimation Table. They have a number of advantages we will discuss below. But when you are operating in short agile delivery cycles, anywhere from 'today' to a week, to 2% - it is then often much easier to *actually try out* an architecture in *practice*, by delivering a subset of it, to the incremental value-delivery process, and *measure and observe* what happens, when that sub-architecture 'hits the fan', with your *real* system, and your *real* stakeholders.

You could study an architecture idea theoretically, for six months, to decide 'accurately' if you wanted to prioritize it. But it might well be faster and cheaper, and a lot more credible, to simply *try it out*, in a delivery cycle, and get answers within a day to a week.

This is not a new idea in scientific research and engineering. That is why they do lots of experiments and prototypes. But I prefer to do things to the *real* system, not an *artificial* prototype. I know how to do that safely, and I only trust **real** systems, to give me the whole truth (almost).

So, for example, when the value-to-cost estimate numbers for architecture options are 'close' to each other, or 'not significantly different', you can, pretty safely, pick *any one of them*, and try it out. Have fun, go with your emotions, at that point.

The tables and numbers will increase the chances, that what you choose, is OK, and that it will probably *not* be a disaster.

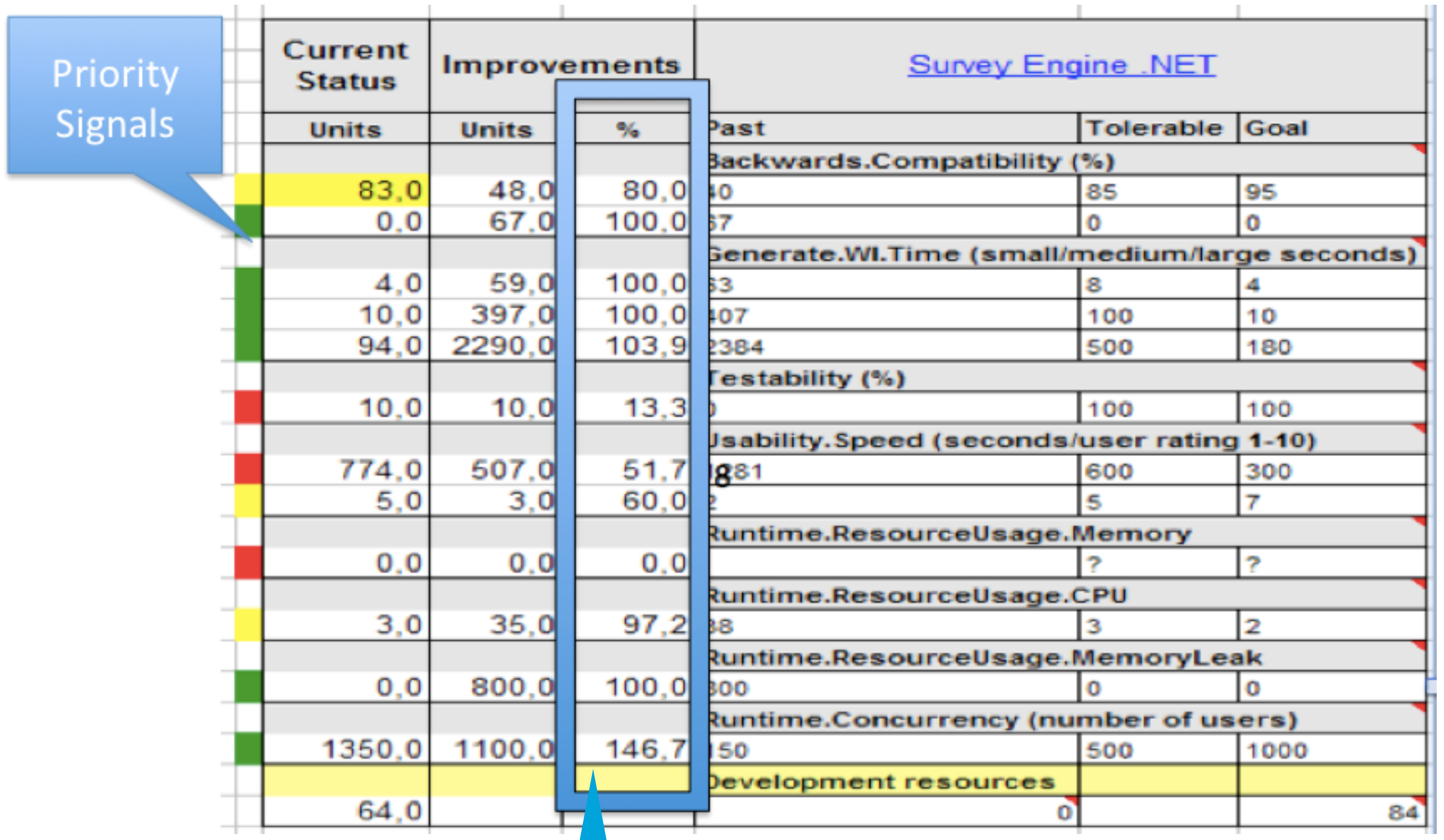
In even very-large projects, we will make a draft Impact Estimation table, with 2 levels of detail (like the Polish ones above in Figure 4.4 B) on a single day. Then we will, same week or next week, *act* on the table, for prioritizing deliveries. We will get *some* value, and a *lot* of learning: and if we are smart, we will be able to do very little harm, and some good.

We can update our estimates, as we get practical measures, 'from our own system' feedback, for use in scaling up, and for implementing similar architectures.

It is also worth noting, that if we prioritize *estimated high values,* in relation to *estimated low cost*s, we should be able to build up a very-rapidly growing set of values, at relatively low costs. That is the experience of my clients, and others like IBM Cleanroom, using essentially the same methods as mine [P2, P3].

5.5 Architecture Prioritization: a business case

Confirmit, Developed 25 vastly improved product qualities every quarter, and blew international competition away.


Priority Signals

	Current Status	Improvements		Survey Engine .NET		
	Units	Units	%	Past	Tolerable	Goal
				Backwards.Compatibility (%)		
🟨	83,0	48,0	80,0	40	85	95
🟩	0,0	67,0	100,0	67	0	0
				Generate.WI.Time (small/medium/large seconds)		
🟩	4,0	59,0	100,0	63	8	4
🟩	10,0	397,0	100,0	407	100	10
🟩	94,0	2290,0	103,9	2384	500	180
				Testability (%)		
🟥	10,0	10,0	13,3	0	100	100
				Usability.Speed (seconds/user rating 1-10)		
🟥	774,0	507,0	51,7	881	600	300
🟨	5,0	3,0	60,0	2	5	7
				Runtime.ResourceUsage.Memory		
🟥	0,0	0,0	0,0		?	?
				Runtime.ResourceUsage.CPU		
🟨	3,0	35,0	97,2	38	3	2
				Runtime.ResourceUsage.MemoryLeak		
🟩	0,0	800,0	100,0	800	0	0
				Runtime.Concurrency (number of users)		
🟩	1350,0	1100,0	146,7	150	500	1000
				Development resources		
	64,0			0		84

Quantified product quality requirements

Figure 5.5 Value Planning at Confirmit. Source [VP, Ch 6.7]. And case [P2]

There is a lot I could explain about the detailed experiences in this real small enterprise case. But see [P2] for that. I will point out one interesting detail here. This % Improvement column, is the 9 out of 12 weeks incremental value-measurement of delivery levels, for 12 values. 100% means 'meeting the competitive Goals', set by Marketing, for release to the international market, after 12 weeks. Notice that after 75% of the time to deadline (9 of 12 weeks) most of the Goal levels are reached (100%) or *surpassed* 146.7%). The average is better than 75% (ahead of the curve, not late). The remaining 25% (3 week cycles) will be used by the 4-person team to attack any value less than 100% (dynamic prioritization). This process ('Evo' was credited officially on their website) was repeatable for the long term. Confirmit quickly got such superior competitive qualities, that they wiped out, and bought up, their international competitors. A Viking Raid indeed. This is 'agile as it should be'. The sub-architecture designs were delegated to the implementation team themselves. We trained these 'genius engineer implementors' for 1 day, in our methods, and left them with our books as supplements. No 'certifications'. Just measurable enterprise results, in their share price. Board members sent us to other of their personal corporate investments. Enterprise architecture, not IT coding. But the coders were designers too. Degreed engineers most of them.

5.6 The Advantages

Dynamic Fact-Based Prioritization has the following advantages:

1. **You can safely launch into delivering a value stream almost immediately, as opposed to *months and years of analysis, design, architecture and other bureaucracy, before results appear*.**

2. **You have constant control over the *profitability* of your project since you are always delivering very high values, at very low costs.**

3. **As you deliver and measure results in small delivery cycles, you are building up facts about everything (architectures, values, costs, stakeholders). So you are making smarter-and-smarter prioritization decisions, based on the new facts.**

4. **The use of tables and numbers, allows you to realistically consider, *multiple values and multiple costs simultaneously*. Only 1 in a million, genius people, can do *that* in their head.**

5. **The use of tables and numbers, and digitization; with *ability to look at the architecture from many levels and perspectives*, makes it *practical and economic* to <u>share the decision-making and prioritization</u> with <u>parallel multi-national teams, over time</u>. The SEA Language model, supported by apps is your Enterprise *Memory and Conscience*.**

6. **As discussed earlier, this dynamic prioritization has demonstrated remarkable ability to succeed, and to not fail. *Quite* different from the failure rates for EA generally [Part 0.0]**

5.6 Architecture Prioritization

Value and cost modeling is helping a startup founder, avoid wasting months, on low-value architecture

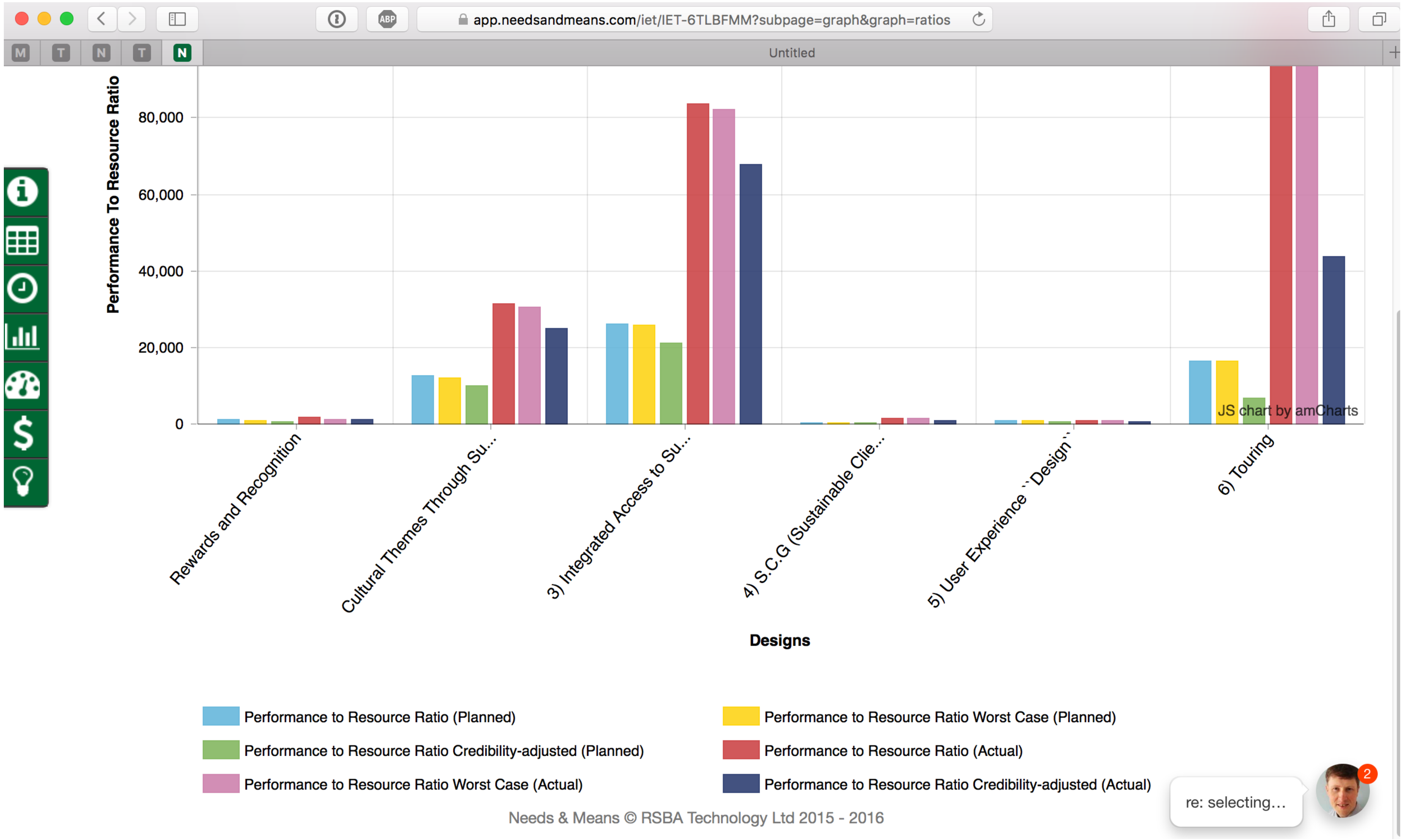


Figure 5.6. Source [VP] Figure 6.5 E Real planning example. A 'bottom line' summary of the estimated impacts of a set of strategies, where the cumulative impact on all top-level critical quantified performance objectives is calculated. Sometimes with respect to the estimated set of budgeted costs. Sometimes with respect to risks (evidence, sources, ± uncertainty ranges) with the strategies. The bar chart is automatically generated from the IE Table information using the Needs and Means tool made by Richard Smith, London [URL73]

Courtesy Incognito Startup Project, Oslo (Gottfried Osei) January 8 2016.

6.0 Architecture Value-Stream Delivery

Constant prioritized flow of measurable values to stakeholders

Basic Concepts

In Systems Enterprise Architecture, we call this 'Value Stream process', '**Evo**'. This is a short form of **Evo-**lutionary Value Optimization. It has a nice acronym too, EVO.

Evo is widely-recognized as the inspiration for agile methods, the most widely-cited is my 1988 book [G15], but I published the ideas much earlier (1972-1976, SEA 4.3). [VA]. See tree chart at end of Part 6.

At least the concept of *small increments*, rather than *big bang*, was picked up by agile-istas. But there was a total failure to pick up my explicit point about **quantified values.** The essential idea is, and was, that the *'small cycles were incrementally driving values towards long-term quantified value-targets'. Nothing new, not even in software [Gilb 15, free download Chapter 15].*

This concept, Evo, can be used to develop the attributes of almost anything, at any scale, including the Enterprise Scale [P14], as proven for example for 20 years, for 21,000 engineers, at Intel.

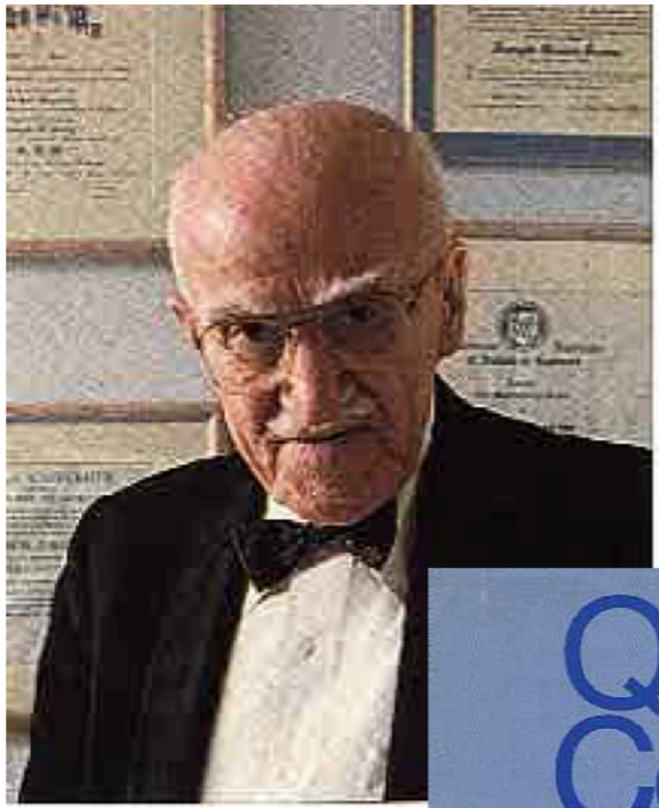
Evo can be used, both on small product-development projects, and on corporate-critical major-flagship corporate-survival product lines, as well as giving a successful quality discipline to 21,000 engineers (you can enterprise architect the *work force*, not just the IT), as it was done at Intel.. We had similar large-scale adoptions at a number of other Corporations, my favorites were HP, and Ericsson. See *these,* and other Corporate Clients, named and anonymous, in many publications, for example [VP, which has detailed HP and Ericsson Case studies in References].

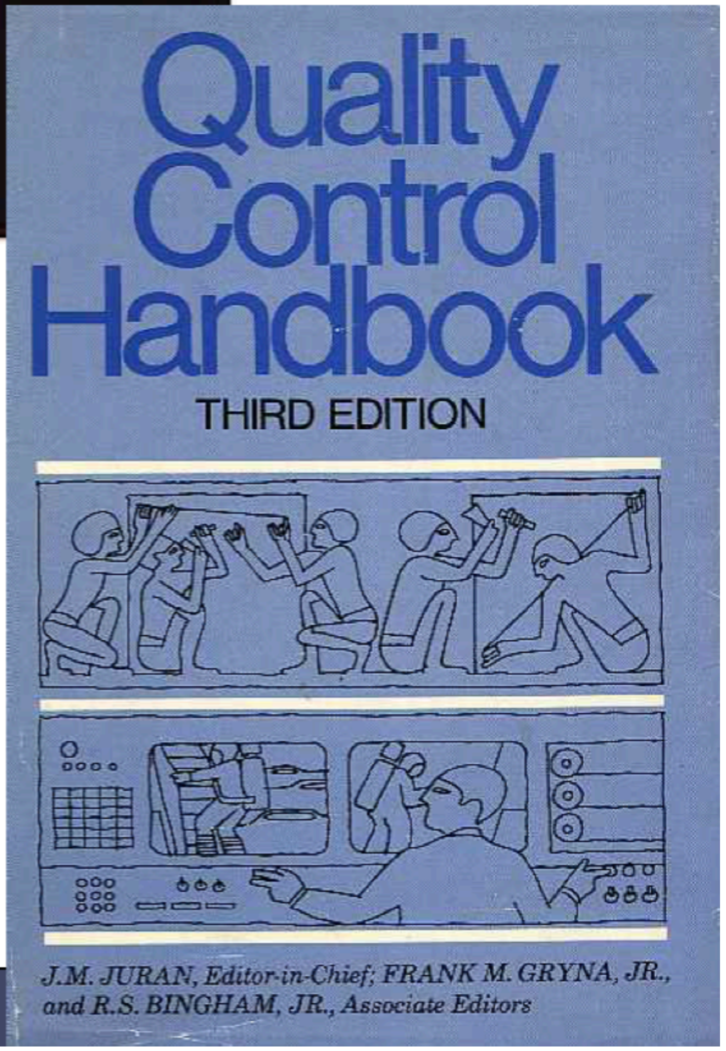
The Evo concept is ancient and simple, and it is built-in to nature: '**work towards long-term survival, and then to success-goals, by constantly adjusting behavior, and your environment ('architecture'), and getting feedback that you are on track**'. That's it, pretty much.

Evo is a cousin of Statistical Process Control, Plan Do Study Act and other smart processes

The Evo Cousins' Commonality

- Learning
- Measurement
- Future Improvement orientation
- Process Improvement
- The Deming/Shewhart (Juran) Statistical ideas
- Eternal learning
- Distinguish between 'chance causes' and 'common causes'
  - fix the common causes.





Quality Control Handbook

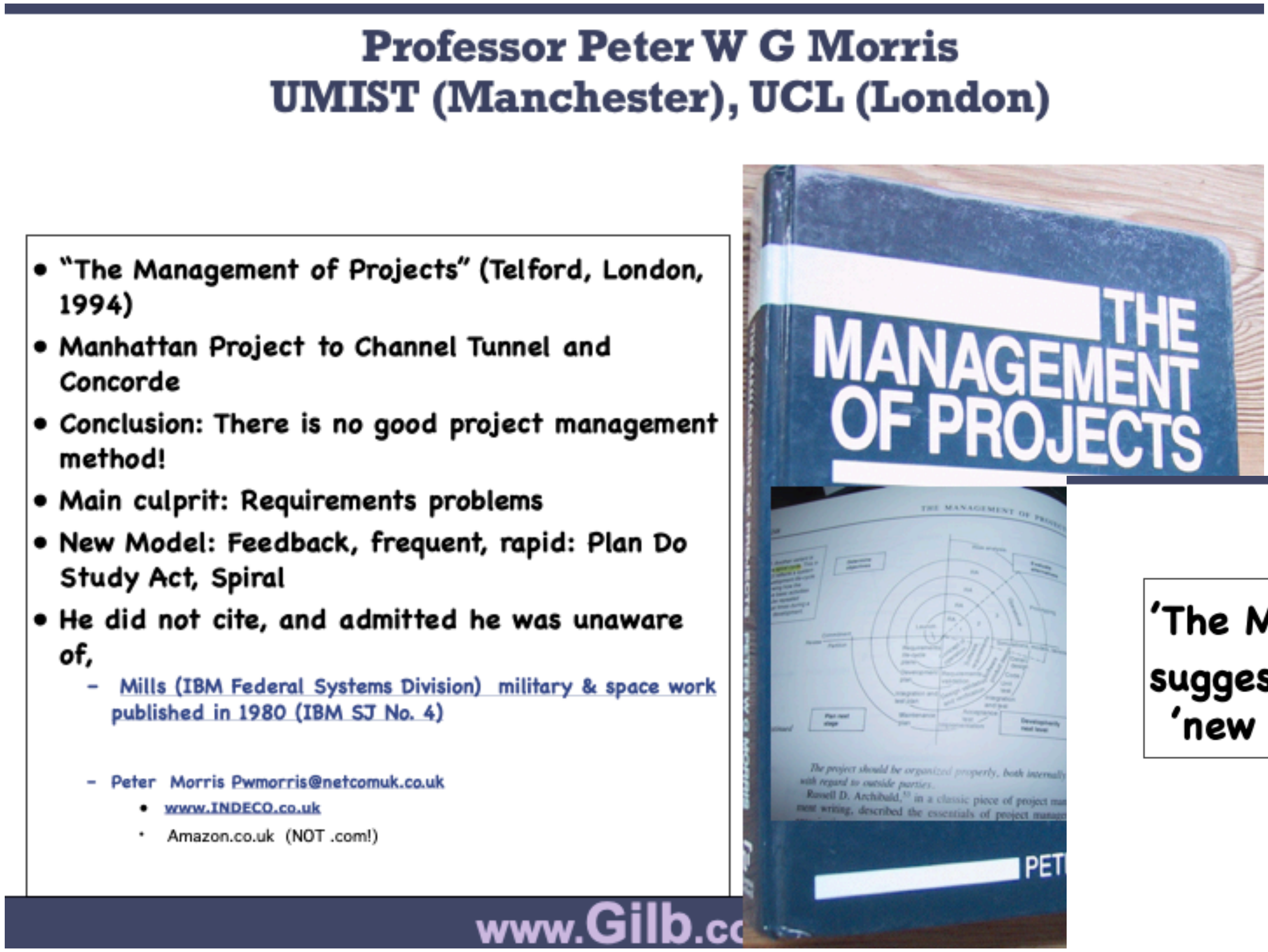
THIRD EDITION

J.M. JURAN, Editor-in-Chief; FRANK M. GRYNA, JR., and R.S. BINGHAM, JR., Associate Editors

www.Gilb.com

Figure 6.0 A. Notice the Egyptian, real, wall carving, showing comparison to a standard, as an engineering process for building the pyramids. Evo is constantly measuring against quantitative value requirements. Value-Driven Architecture.

And (this was 1994, well before Agile was popular) concluded that the new project management model had to be more iterative. He later worked with Project Management Instituted

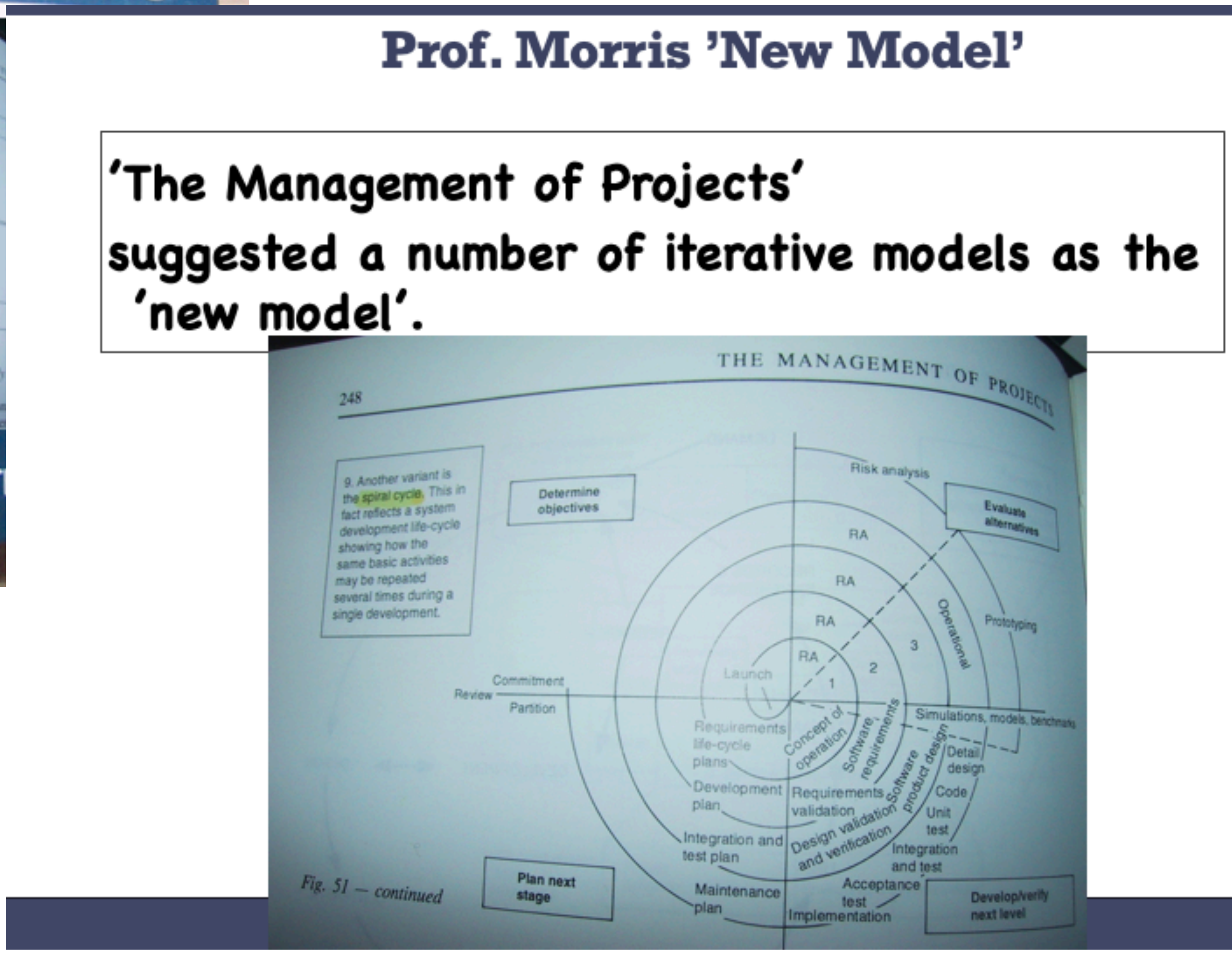


Professor Peter W G Morris
UMIST (Manchester), UCL (London)

- "The Management of Projects" (Telford, London, 1994)
- Manhattan Project to Channel Tunnel and Concorde
- Conclusion: There is no good project management method!
- Main culprit: Requirements problems
- New Model: Feedback, frequent, rapid: Plan Do Study Act, Spiral
- He did not cite, and admitted he was unaware of,
  - Mills (IBM Federal Systems Division) military & space work published in 1980 (IBM SJ No. 4)
  - Peter Morris Pwmorris@netcomuk.co.uk
    - www.INDECO.co.uk
    - Amazon.co.uk (NOT .com!)

Figure 6.0 B. Failed large complex projects is not special for IT. And better architecture itself is not the solution noted. It is better requirements and small increment feedback!

An iterative model. Spiral



Prof. Morris 'New Model'

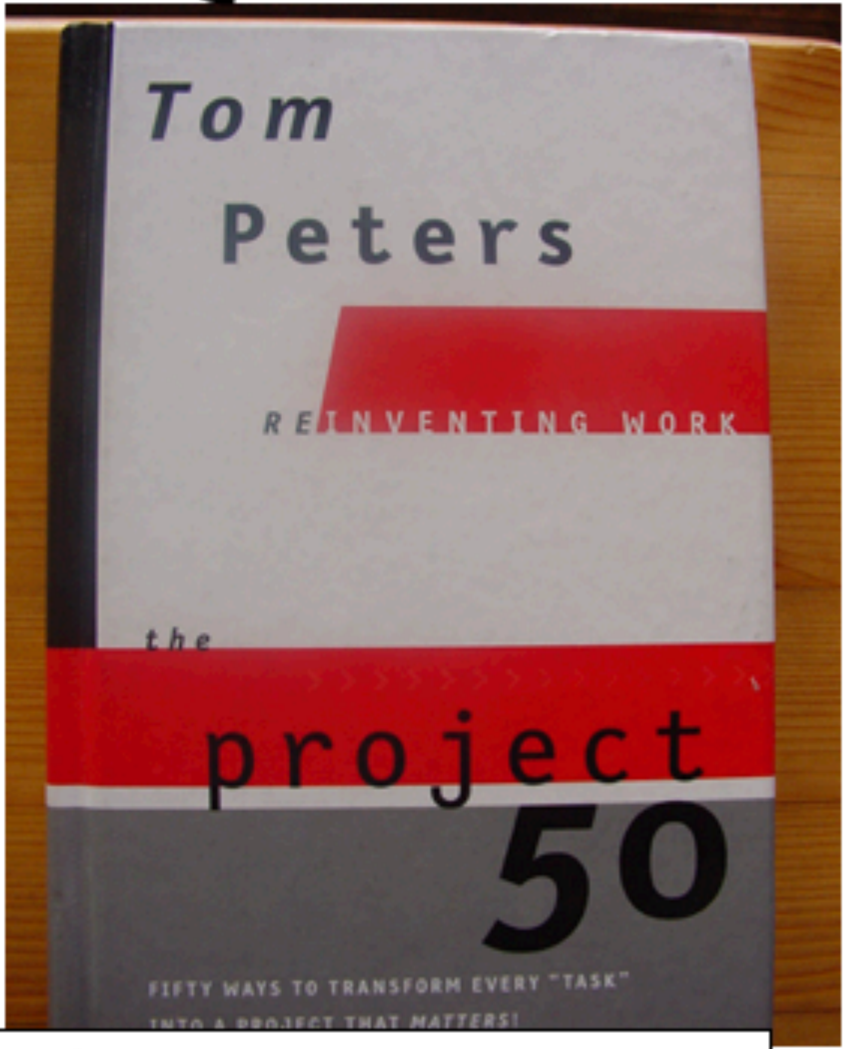
'The Management of Projects' suggested a number of iterative models as the 'new model'.

THE MANAGEMENT OF PROJECTS
248
9. Another variant is the spiral cycle. This in fact reflects a system development life-cycle showing how the same basic activities may be repeated several times during a single development.
Determine objectives
Risk analysis
Evaluate alternatives
RA
Operational
Prototyping
Launch
Commitment
Review
Partition
Requirements life-cycle plans
Concept of operation
Software requirements
Simulations, models, benchmarks
Detail design
Code
Unit test
Development plan
Requirements validation
Software product design
Design validation and verification
Integration and test plan
Integration and test
Acceptance test
Maintenance plan
Implementation
Plan next stage
Develop/verify next level
Fig. 51 — continued

A page from the book

The Best-Selling Management Author and Corporation analyst Tom Peters has long observed that rapid practical iterations is the way to success in the enterprise. See also Steve Blank for same emphasis https://steveblank.com

Quick Prototyping á la Peters


Tom
Peters
REINVENTING WORK
the
project
50
FIFTY WAYS TO TRANSFORM EVERY "TASK" INTO A PROJECT THAT MATTERS!

Tom Peters

Reinventing Work, the project 50. Alfred A. Knopf, New York, 2000, ISBN 0-375-40773-1. See Peters' website www.tompeters.com, $15.95

See also his book 'the Quick Prototype50'.

See especially his emphasis on 'quick prototyping' in relation to Evolutionary project management.

A.S.A.P.I.N.S.

As Soon As Possible If Not Sooner

"1. Now. Right now. Take some little - tiny! - element of your project. Corral a surrogate customer. Talk to him/her about it. That is … test it. Now.

2. Your immediate goal: "Chunk up" the next three weeks. I.e.: Define a set of practical micro-bits … that can be subjected to real-world tests..

Observation: There is *no* situation - even at Boeing - where you cannot concoct a sorta-real-world-micro-test of some piece of your project .. Within a few hours to two or three days.

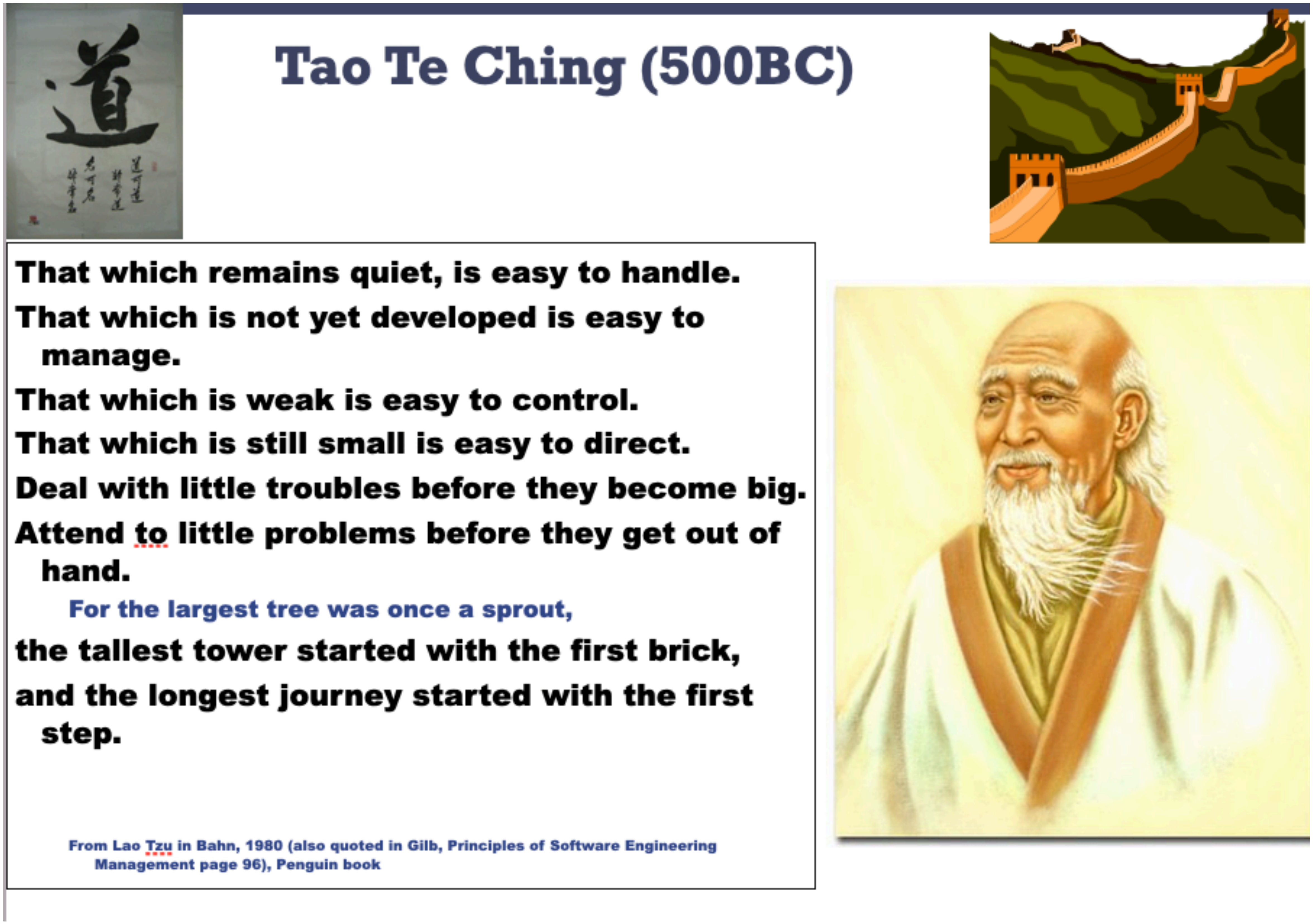
Quick Prototyping Excellence = Project Implementation Excellence. (No kidding… it's almost that basic!)"

Pages 138-9

X

Figure 6.0 C. *Source: Decomposition by Value slides MASTER MAY2016*

The Evo ideas are not new.



Tao Te Ching (500BC)

That which remains quiet, is easy to handle.

That which is not yet developed is easy to manage.

That which is weak is easy to control.

That which is still small is easy to direct.

Deal with little troubles before they become big.

Attend to little problems before they get out of hand.

For the largest tree was once a sprout,

the tallest tower started with the first brick,

and the longest journey started with the first step.

From Lao Tzu in Bahn, 1980 (also quoted in Gilb, Principles of Software Engineering Management page 96), Penguin book

Figure 6.0 D. Source: [G15]. And Evo Tutorial Master, slide 59

Design Components increment to the larger architecture

The evo basic idea, as simply put as I can.

**‘work towards long-term survival,
and then to success-goals,
by constantly adjusting behavior,
and your environment (‘architecture’),
and getting feedback that you are on track’.**

Figure 6.0.1 E. Re Quote the basic idea of Evo from SEA 6.0, here above.

Evolutionary Project Management **309**

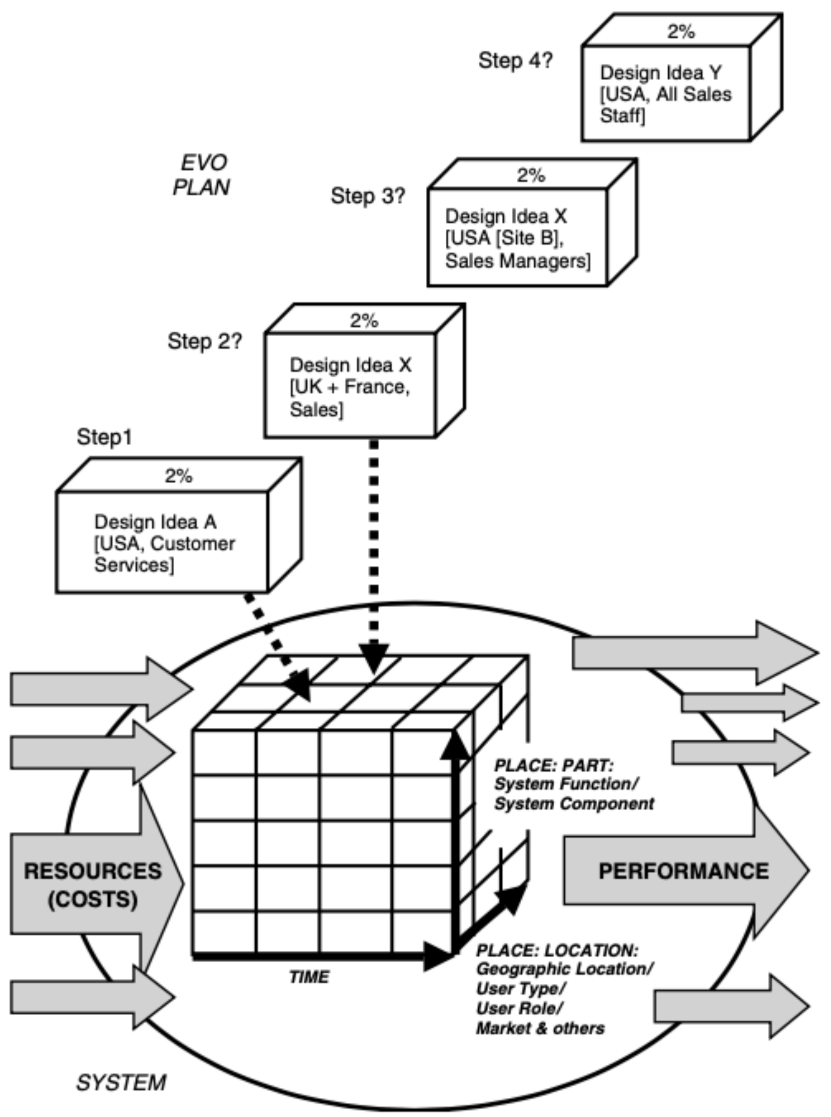

EVO PLAN
Step 4?
2%
Design Idea Y [USA, All Sales Staff]
Step 3?
2%
Design Idea X [USA [Site B], Sales Managers]
Step 2?
2%
Design Idea X [UK + France, Sales]
Step1
2%
Design Idea A [USA, Customer Services]
RESOURCES (COSTS)
PERFORMANCE
PLACE: PART: System Function/ System Component
PLACE: LOCATION: Geographic Location/ User Type/ User Role/ Market & others
TIME
SYSTEM

Figure 10.4
An Evo plan and the system: the diagram shows the steps being sequenced for delivery. Each step delivers a set of performance attributes (a subset of the long-term planned results), and consumes a set of resources (a subset of the long-term budgets), in a specific place {location, system component} and at a specific time (for delivering the benefits). The purpose of this diagram is to show that each Evo Step will become a sub-component of the evolving system's long-term vision and plan.

Figure 6.0. E. *From [CE, p. 309]*

The detailed tactics of Evo

1. frequent delivery of system changes (steps)
2. steps delivered to stakeholders for real use
3. feedback obtained from stakeholders to determine next step(s)
4. the existing system is used as the initial system base (never build a new system!). See next 6.1 pages for detail.
5. small steps (ideally between 2%-5% of total project financial cost and time)

7. ste cost delivery

The Result Cycle for an Evo Step

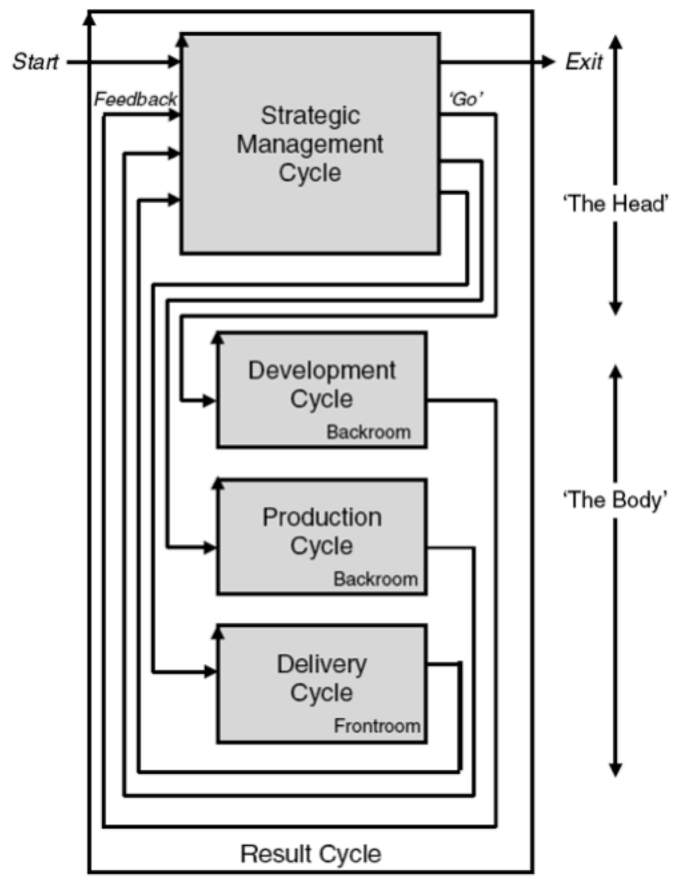


Figure 6.1 source [CE, p. 306]

8. feedback used 'immediately' to modify long-term plans and requirements and, also
9. ..to decide on the next-step total-systems approach ('change anything that helps') -
10. results-orientation ('delivering the results' is prime concern)

6.1.1 Architecture Value-Stream Delivery: Evo, from existing system base

4. the existing system is used as the initial system base (never build a new system!)

The above Evo tactic, 4., is worth a separate page of explanation and justification.

It seems to rarely be asserted or discussed, in the agile, architecture, or project-management literature.

Make no mistake, this is a major Enterprise Architecture option ('build from current system', or 'build new from scratch').

I have practiced this since 1960 in all my projects, and in all my client advice, happily. No exceptions. Just common sense.

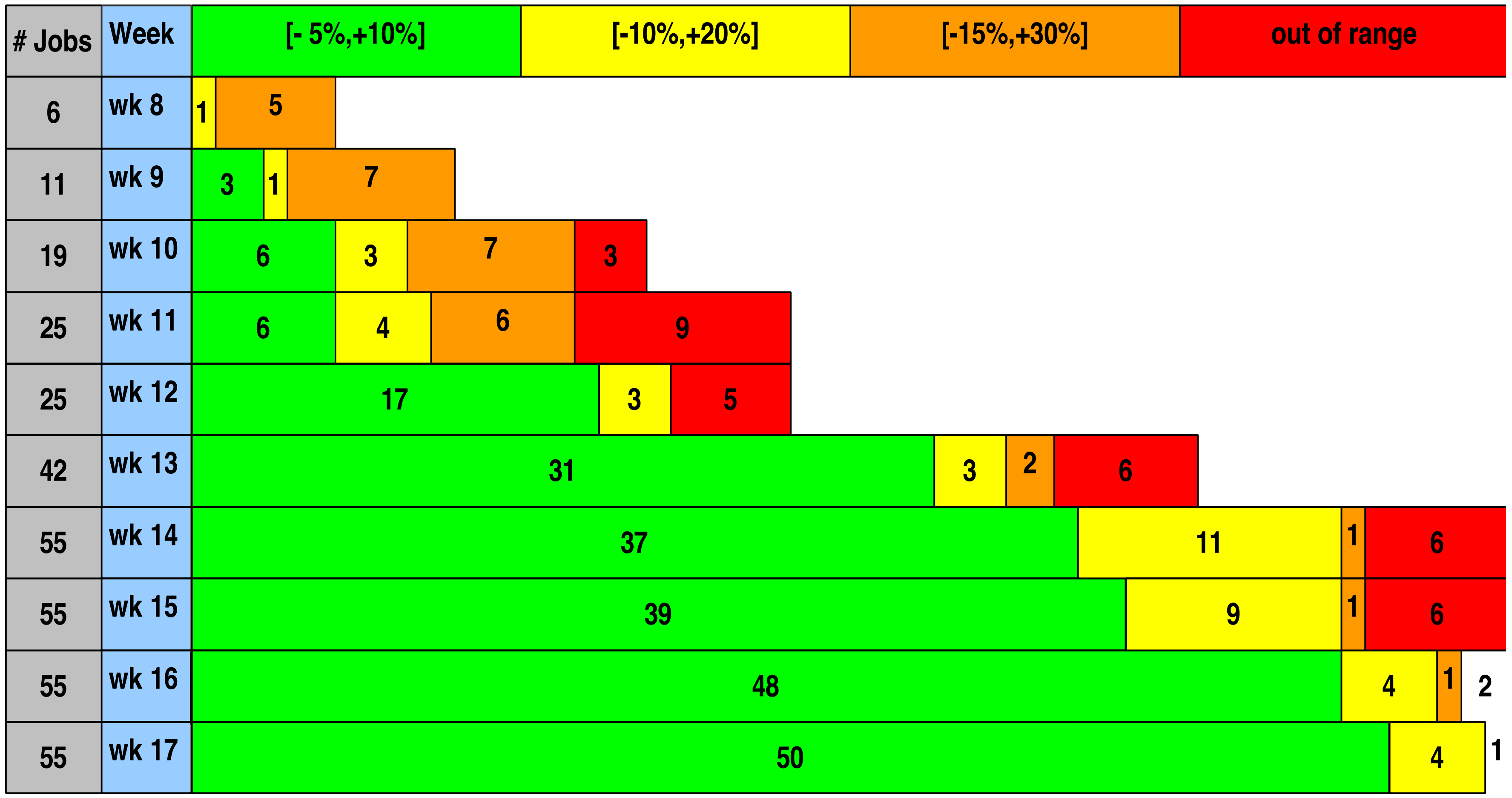
But if I look at practice around me, such as large government projects, they are always building big-bang new systems, and never 'architecting to deliver incremental (like monthly) value' measurably to the existing system. Then they always put considerable pride in being 'agile' by building the new code using Scrum or SAFe. They are also failing catastrophically in public, and they do not seem to understand why. This is why, the architect did not adopt the Evolutionary Value Delivery architecture at the top-level of architecture, and then architect to find, and deliver, increments of sub-architecture, moving us towards the Critical Stakeholder Value Targets.

Here are the advantages of real 'Evo': delivering to the 'old system'

1. The old system, whatever its flaws and problems and age, exists now, and is operating now. It cannot suddenly be recreated on new platforms.
2. The 'old system' has a great deal of practical knowledge in it, and surrounding it, for users. This is near impossible to analyze and re-create correctly.
3. If you focus efforts on improvements, stakeholder values, that are critical: and build these into the existing system, you are far more likely to get the desperately needed improvements this year, instead of a public shame failed big bang project in 5 to 8 years.
4. Incremental changes can be inserted now, in such a way that they are relatively portable, to any later new underlying technology architecture we might insert later. This is not least an architecture planning concern, to make sure this is so.
5. If your architects claim they cannot do this, I claim they are incompetent, uninformed, and unimaginative: and you need better architects.
6. A good analogy is cities, and buildings. We do not wipe out or abandon London, with all its strange attributes: and build a new London elsewhere. Even single buildings, think Buckingham Palace, Houses of Parliament; are improved, but not wiped out, and replaced.
7. If your architects cannot even succeed in delivering incremental architecture, and consequent incremental value in the short term, on a small scale; then they are surely incompetent to rebuild the system anew big bang.

6.1.1 Architecture Value-Stream Delivery: Evo, from existing system base

Value Delivery Increments in practice, to an existing system



# Jobs	Week	[- 5%,+10%]	[-10%,+20%]	[-15%,+30%]	out of range
6	wk 8		1	5	
11	wk 9	3	1	7	
19	wk 10	6	3	7	3
25	wk 11	6	4	6	9
25	wk 12	17	3		5
42	wk 13	31	3	2	6
55	wk 14	37	11	1	6
55	wk 15	39	9	1	6
55	wk 16	48	4	1	2
55	wk 17	50	4		1

Figure 6.1.1 A. Source Evo Tutorial MASTER 1 day

Frank van Latum,
The Manager

The GxxLine PXX Optimizer EVO team proudly presents the success of the Timing Prediction Improvement EVO steps.
Shown are the results of the test set used to monitor the improvement process.
The size of the test set has grown, as can be seen in the first column. (In the second column the week number is shown.)
We measured the quality of the timing prediction in percentages, in which –5% means that the prediction by the optimizer is 5% too optimistic.
Excellent quality (–5% to +10%) is given the color green, very good quality quality is yellow, good quality is orange, & the rest is red.
The results are for the ToXXXz X(i) and EXXX X(i), and are accomplished by thorough analysis of the machines, and appropriate adaptation of the software.
The GXXline Optimiser Team presented the word document below to the Business Creation Process review team.

The results were received with great applause. The graphics are based on the timing accuracy scale of measure that was defined with Jan Verbakel.

Ignorance about value agile

Here are the false understandings that inhibit people from delivering to the old systems.

1. **They are afraid of disturbing a fragile old system with lots of technical debt. (think every change creates 3 bugs).**
   - If your system is really precarious, you, by definition, have 'a critical value-set related to safe changes' like 'Stability' and you are not managing them with architecture.
   - You can set quantified quality objectives for maintainability, and availability, for example; and systematically evolve the system. (Confirmit [S14] did exactly this, see [CE] for 'how generally', Chapter 5)
2. **They do not have qualified staff who understand the older technology, or indeed who even want to continue using the older technology (think COBOL).**
   - There are lots of people, who for the right pay and conditions will be happy to learn and master old systems.
3. **Architects have no competence in analyzing and specifying quantified stakeholder-value requirements. So they are not 'culturally enabled' to think in terms of, for example, increments of security, or increments of stability.**
   - Retrain or remove! See Parts 1, and 10 of this book).
   - Architects who cannot do this are 'seriously incompetent' to deal with the old system, and worse they will not be able to design in qualities of this kind, to future systems. Future systems will be fragile, unstable and full of technical debt. Bad architecture last time, is the main reason for your fragile systems today.
   - See [S14] for Confirmit and other experiences, and how to quantify technical debt in practice.
4. **The architects are not trained and do not know, and do not 'know that they do not know' how to usefully decompose big architecture ideas, into a value-stream of sub-architectures (see Part 4)**
   - You are hold the training manual in your hands. Train them to decompose into bite-sized value-delivery steps. Do not 'let loose' on your enterprise, megalomanic architects.
   - Ignorance with Architecture Certification is still ignorance, and it is no excuse. It is 'certifiably' insane.
5. **Management (think CTO level) is no wiser, and has the illusion that the certified architects know what they are doing. They don't, in my opinion. But the boss feels they have to believe the counterarguments for 'why we have to build a new system from scratch'.**
   - Ask your architects some simple questions [12?] like. 'Show me how fast and how well-proven, your last architecture assignment, actually delivered stakeholder value to the system. Or come back when you have some useful experience'.
6. **The 'digitalization' workforce and suppliers (think, large consultancy bodyshops) are delighted to take your money for billions, and for years, without ever having to make real improvements at all. In Norway, Parliament has had to intervene, and the Press (Akson 2020) to stop the corrupt madness. And corruption is what they call it. Are you corrupt, or just ignorant?**
   - Stop using these techie terms. (Digitization, Agile) Start calling it the Critical Rapid Value-Improvement Program, and have the results quantified at all times. Report to the government or Board what your results are ! [S12, ICL Case]. Managers and architects should do this.

6.1.1 Architecture Value-Stream Delivery: Evo, from existing system base

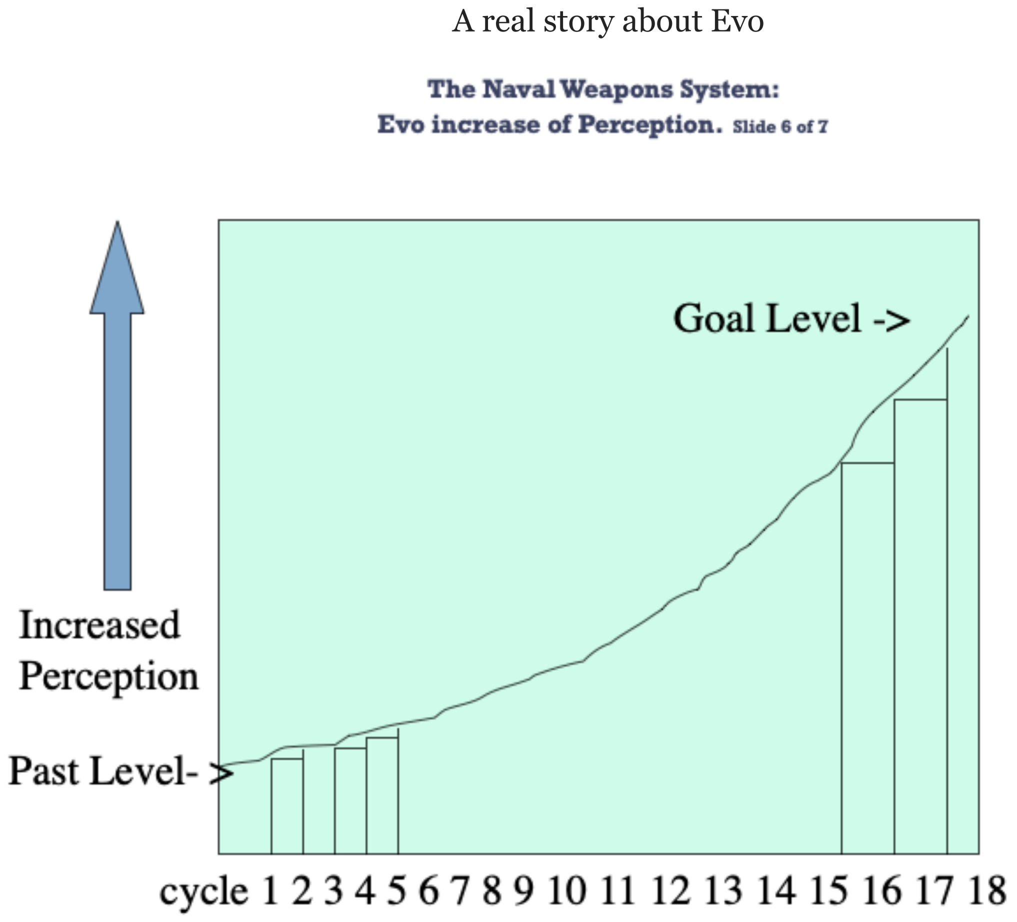


Figure 6.1.1 B. Source: Gilb slides, The craft perception value increases as we add craft templates in priority sequence.

Once, when holding a public course
on the EVO method in London,
a participant came to me in the first break (He was head of a Naval Research Lab)

and said he did not think he could use my 'Evolutionary method'.

Why?
"Because my system is to be mounted on a new ship not destined to be launched for three years."

The Barrier:
"It cannot be done until the new {thing, building, organization, system}.... is ready in some years time".

Did you notice I applied the Backroom/Frontrrom concept, to solve the problem? [6.1.4]

Faith:

I did not know anything about his system, at that point. But I expressed confidence that there is always a solution, and bet that we could find one during the lunch hour.

The Case:

He started our lunch by explaining that his weapons research team made a radar-like device that had two antennas instead of the usual one, which had their signals analyzed by a computer before presenting their data. It was for ship-and-air traffic, surrounding the ship it was on.

The Shift of attention:

I made a stab at the "value results" he was delivering, and who his "stakeholder" was, two vital pieces of insight for making Evolutionary delivery plans.

"May I assume that the main value you provide is "increased accuracy of perception", and that your "stakeholder" is Her Majesty's Navy?"

"Correct." He replied.

"Does your 'box' work more or less, now, in your labs?", I ventured. (Because if it did, that opened for immediate use of some kind)

"Yes", he replied.

"Then what is to prevent you from putting it aboard one of Her Majesty's current ships, and ironing out any problems in practice, enhancing it, and possibly giving that ship increased capability in a real war?" I tried, innocently.

(The sub-architectures are to put profile data aircraft-by-aircraft, in priority sequence into the system)

"Nothing!", he replied. And at that point I had won my bet, 20 minutes into the lunch.

Notice the "method" emerging from this example:

1. Identify the real stakeholder,
   and plan to deliver results to them.
2. Identify the real improvement results
   and focus on delivering those results to the real stakeholder.

in other words:

1. Do not get distracted by intermediaries (the new ship)
   think (other stakeholders) "The Royal Navy" or even "The Western Alliance".
2. Do not get distracted by the perceived project product (the new radar device for the new ship):
   think "increased accuracy of perception".

6.1.2 Architecture Value-Stream Delivery: The Evo Process

The Competitive Engineering book formal definition of Evo, with the Front End, Strategic Management Cycle. See Later the essentially same front end as the Evo Project Startup Week, which is a practice we evolved.

The Evo process standard

The 'Evo' (**Evol**utionary) Method for Project Management.

Process Description

1. Gather from all the key stakeholders the top few (5 to 20) most critical goals that the project needs to deliver.
Give each goal a reference name (a tag).

2. For each goal, define a scale of measure and a 'final' goal level.
For example: *Reliable: Scale: Mean Time Before Failure, Goal: 1 month.*

3. Define approximately 4 budgets for your most limited resources
(for example, time, people, money, and equipment).

4. Write up these plans for the goals and budgets
(*Try to ensure this is kept to only one page*).

5. Negotiate with the key stakeholders to formally agree the goals and budgets.

6. Plan to deliver some benefit
(that is, progress towards the goals)
in *weekly* (or shorter) increments (Evo steps).

7. Implement the project in Evo steps.
Report to project sponsors after each Evo step (weekly, or shorter) with your best available estimates or measures, for each performance goal and each resource budget.

On a single page, summarize the *progress to date* towards achieving the goals and the costs incurred.

8. When all Goals are reached: 'Claim success and move on'
a. Free remaining resources for more profitable ventures.

Figure 6.1.2 A. Source and more detail *http://www.gilb.com/DL487* *also from [CE] Ch.10*

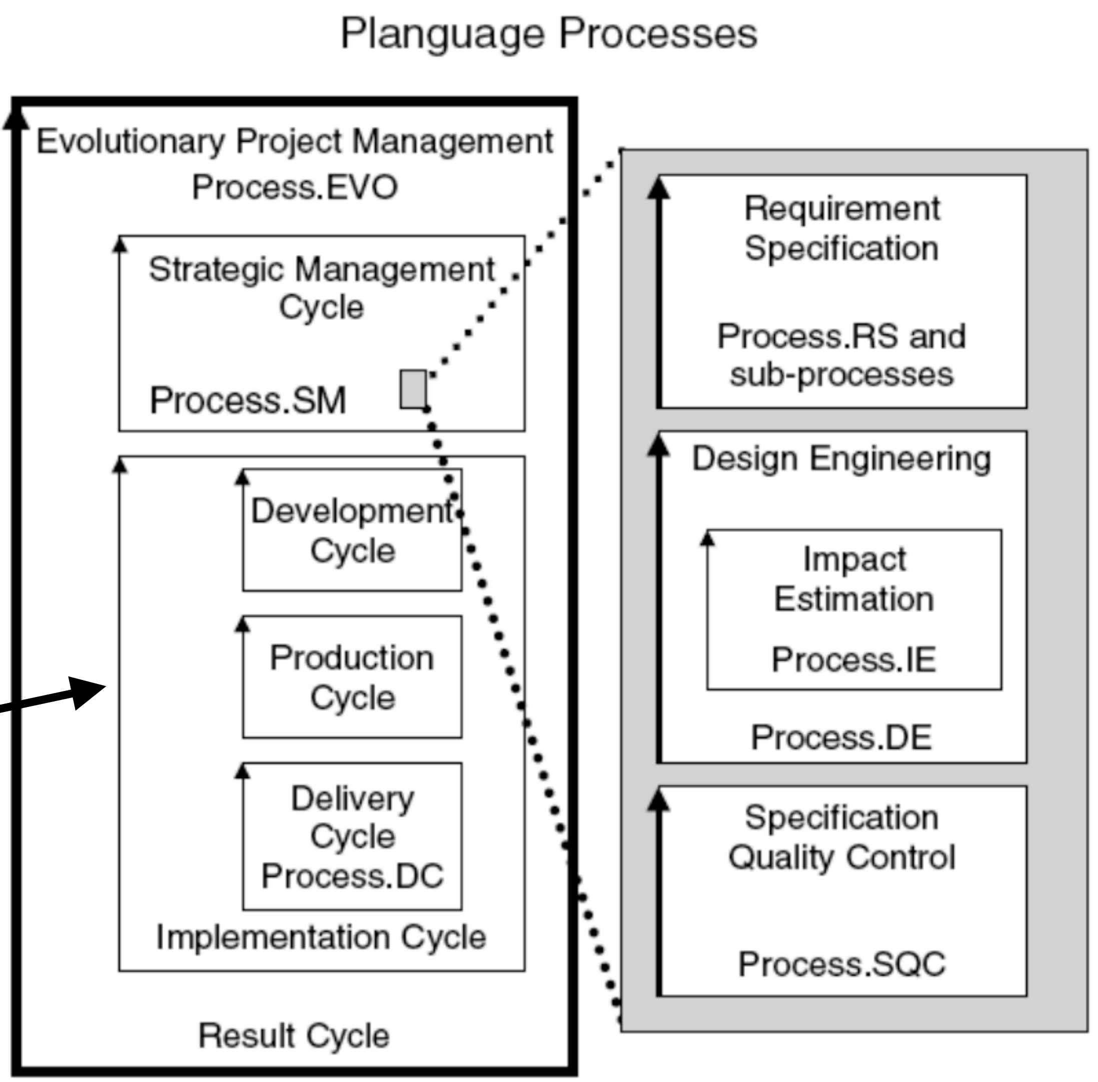

Planguage Processes
Evolutionary Project Management Process.EVO
Strategic Management Cycle
Process.SM
Development Cycle
Production Cycle
Delivery Cycle Process.DC
Implementation Cycle
Result Cycle
Requirement Specification
Process.RS and sub-processes
Design Engineering
Impact Estimation
Process.IE
Process.DE
Specification Quality Control
Process.SQC

Figure 6.1.2 B. Source [CE, 1.3]
Note the rectangle with arrow is the Planguage icon for a 'process'

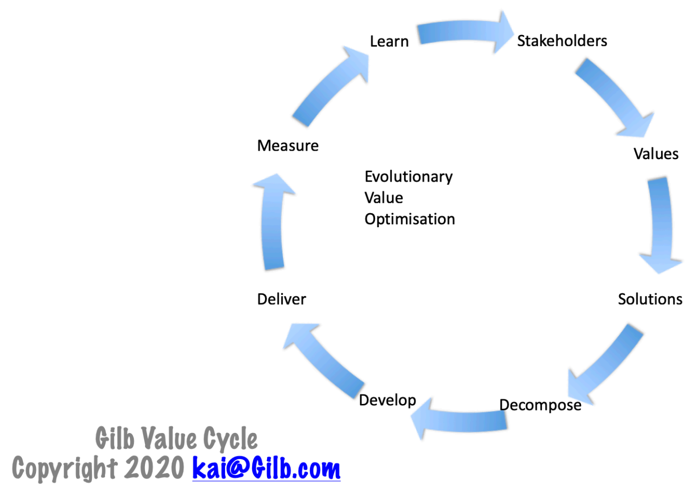
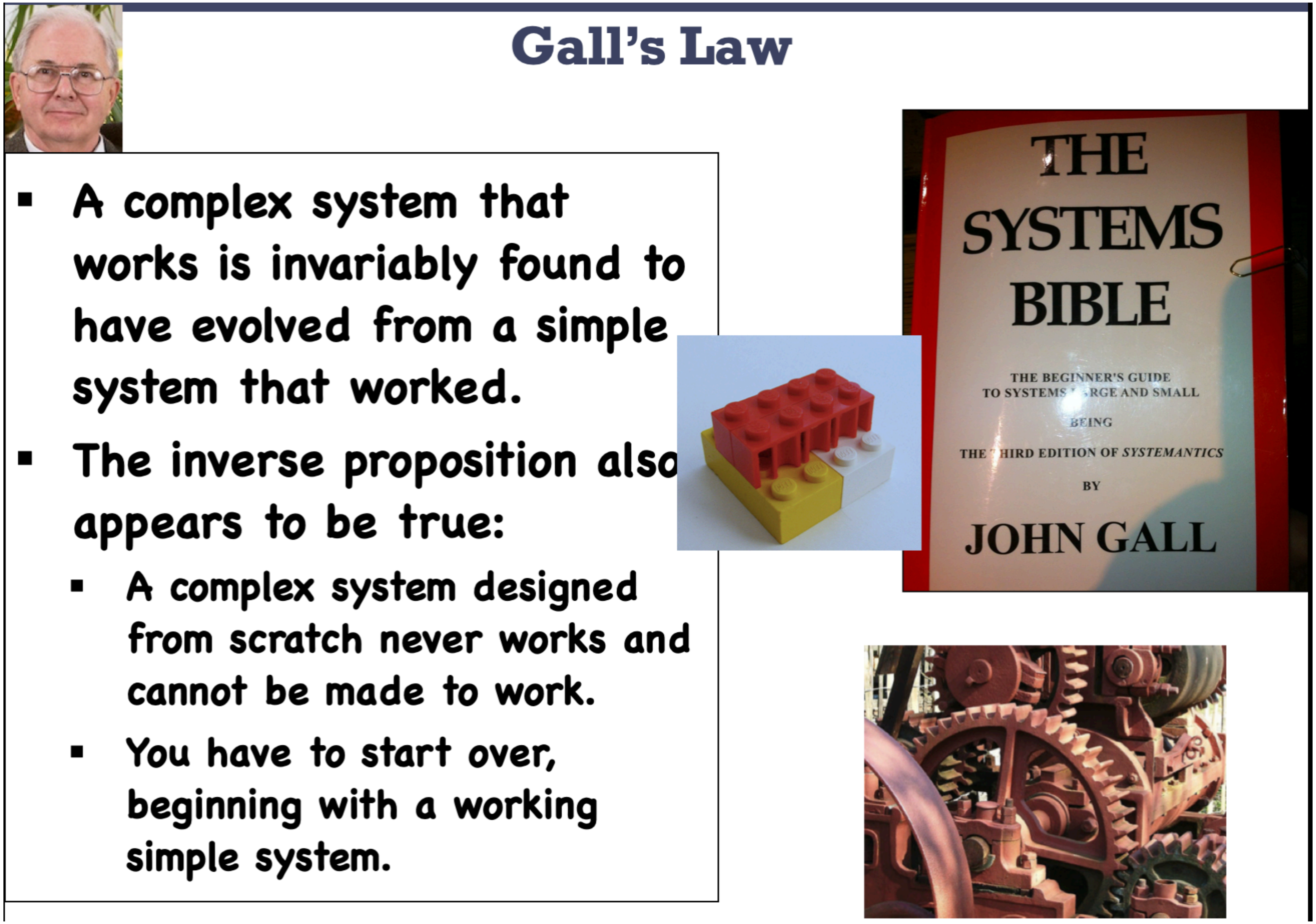


Figure 6.1.2 C. The Gilb Evo Cycle

If you evolve in small steps, you can work out the bad interactions, one by one. If you do too much at once, you get far too many bad interations, and cannot understand or deal with them, which leads to failure.



Gall's Law

- A complex system that works is invariably found to have evolved from a simple system that worked.
- The inverse proposition also appears to be true:
  - A complex system designed from scratch never works and cannot be made to work.
  - You have to start over, beginning with a working simple system.

THE SYSTEMS BIBLE
THE BEGINNER'S GUIDE TO SYSTEMS LARGE AND SMALL
BEING
THE THIRD EDITION OF SYSTEMANTICS
BY
JOHN GALL

Figure 6.1.2 D. *Source: Evo Tutorial Slides MASTER 1 Day 2011*

How much planning up front?

The ‘design sprint’ [VR] has become a popular idea for starting small projects, like product website. I am sure it is good for simpler projects, and simpler people. But I could never like it or do it, because it knows nothing about quantifying critical values and costs. Quantification is absolutely essential for serious large-scale and complex systems.

I have for a long time (since at least 1990 or earlier) had something similar, in the sense of ‘a week to start up a project’. But my startup week is suitable for very large and complex enterprise architecture efforts.

It starts on the first day, with quantifying the top-10 critical stakeholder objectives, as requirements. We do the best we can on day one. Often several parallel teams work on a few requirements. But at end of day one, we deliver a one-page summary of the quantified objectives.

And we do our architecture, the top-10 architecture ideas, to meet those targets, on the second day. A one-page summary of the architecture is the output of day 2.

The third day, we evaluate the architecture, as best we can, against the objectives and costs.

The fourth day, we decompose, and primarily try to find a next-week practical delivery-step, and try to deliver a real improvement, to a real system.

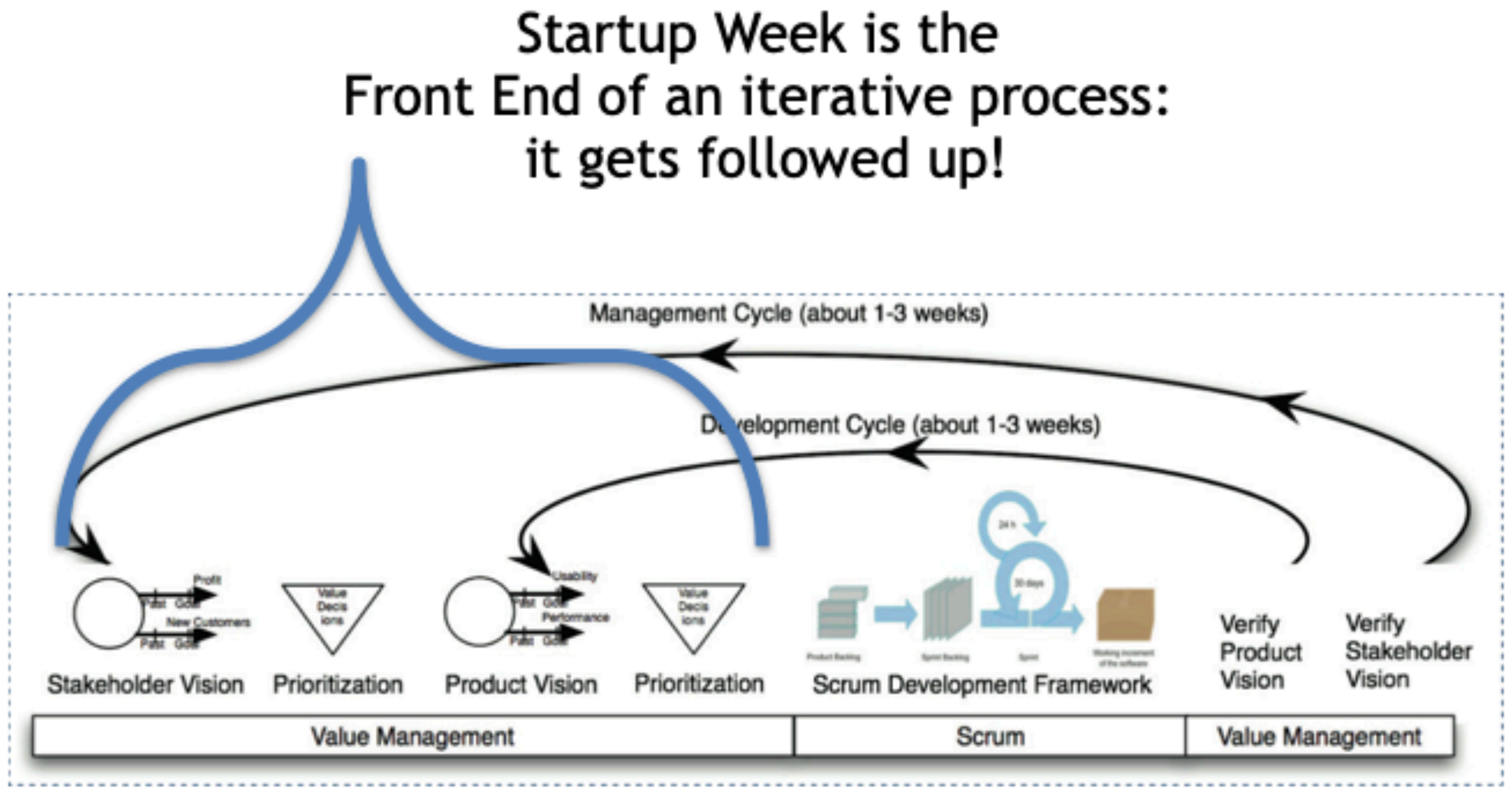
The last day, we present the ideas to management for approval, to try to deliver value, next week.

At McDonnell Douglas Aircraft (Now Boeing) we did this for 25 aircraft projects, 5 each week in parallel [S8]. Nothing to do with IT. The US DoD Project was an Army Personnel IT system.[http://www.gilb.com/DL451].

All this succeeded, as far as I can track them. Never heard a problem, and got plenty of written praise from top management {See references].

6.1.3 EVO STARTUP PROCESS: The 1st Week

We quantify, structure, and are very clear about the most critical requirements, stakeholders and architecture, for a week, using Evo. Then we dive in and 'just do it' in cycles, until we reach requirements targets, or run out of resources.



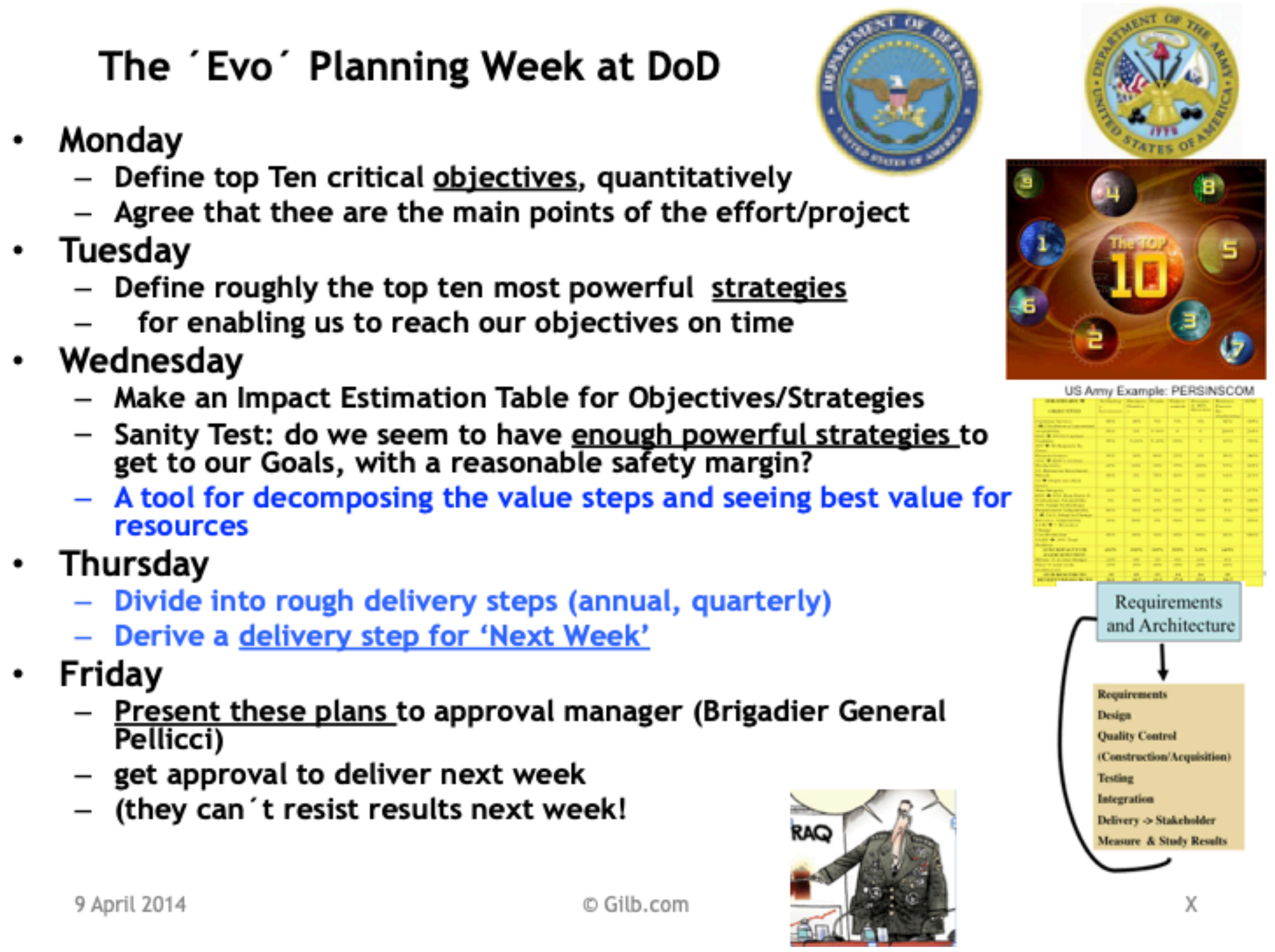
Startup Week is the
Front End of an iterative process:
it gets followed up!

Figure 6.1.3 A. Compare this to Fig. 6.1 B

You can think of this as a smart front-end to Scrum. But Scrum itself has to add it to the Scrum framework. Add quantified value-and-cost management. It is not there in the Scrum framework. Jeff Sutherland has publicly recommended these ideas [VA].

6.1.3 EVO STARTUP PROCESS: The 1st Week

The Startup Week as practiced at US Dept of Defense, to save 8 year old system, which failed in Iraq War I



The ´Evo´ Planning Week at DoD

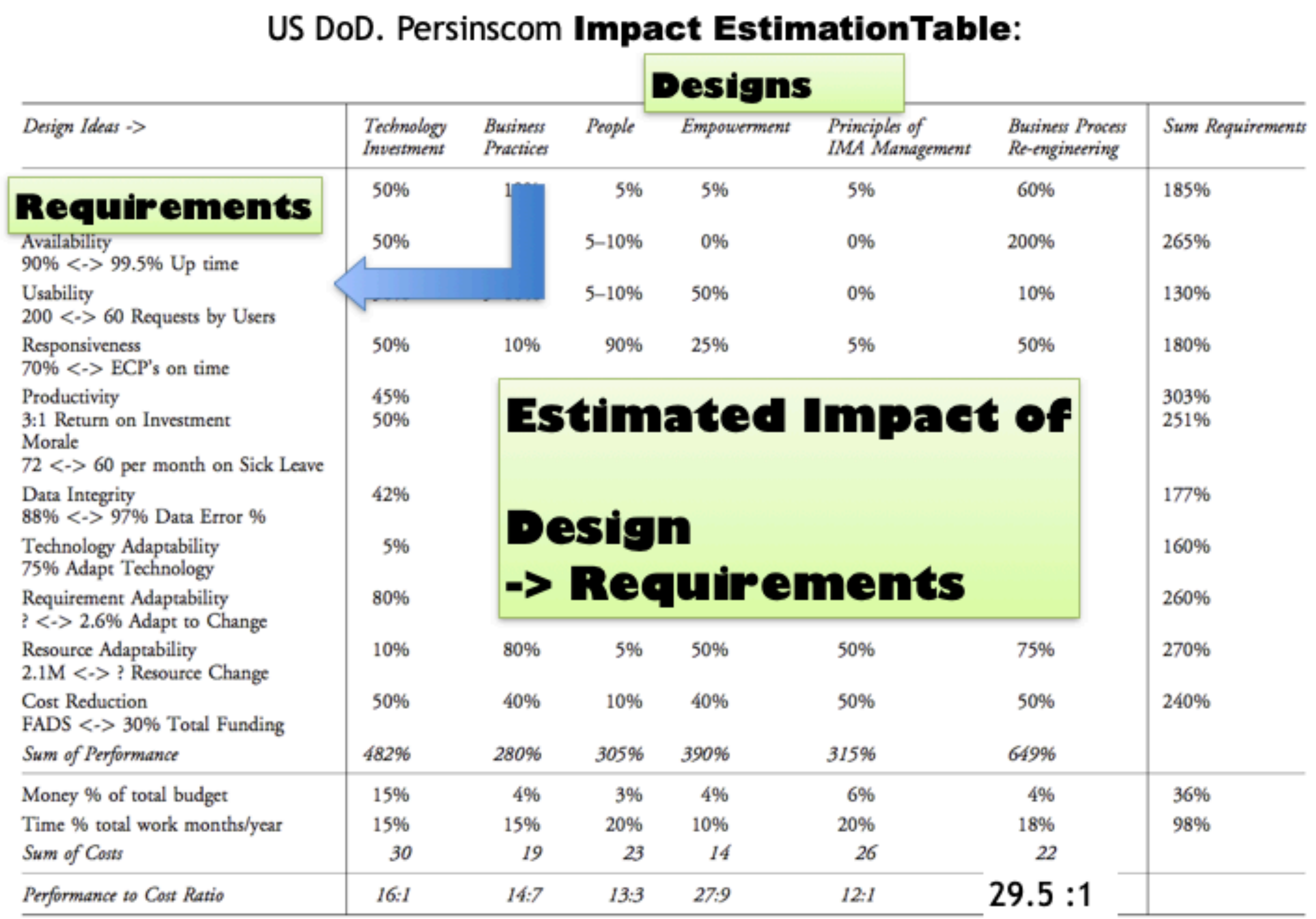
- Monday
  - Define top Ten critical <u>objectives</u>, quantitatively
  - Agree that thee are the main points of the effort/project
- Tuesday
  - Define roughly the top ten most powerful <u>strategies</u>
  - for enabling us to reach our objectives on time
- Wednesday
  - Make an Impact Estimation Table for Objectives/Strategies
  - Sanity Test: do we seem to have <u>enough powerful strategies</u> to get to our Goals, with a reasonable safety margin?
  - A tool for decomposing the value steps and seeing best value for resources
- Thursday
  - Divide into rough delivery steps (annual, quarterly)
  - Derive a <u>delivery step for 'Next Week'</u>
- Friday
  - <u>Present these plans</u> to approval manager (Brigadier General Pellicci)
  - get approval to deliver next week
  - (they can´t resist results next week!

9 April 2014 © Gilb.com X

Figure 6.1.3 B. Source: An Agile Project Startup Week slides.
Also: 111111 Unity Method of Decomposition into weekly increments of value delivery.
Case Study US Dept. of Defence. (10 min slides). http://www.gilb.com/DL451

6.1.3 EVO STARTUP PROCESS: The 1st Week

Real 3rd Day evaluation of the Architecture impacts on the targets and costs.
The commanding General said it was the best planning method he had ever seen, and he "went to West Point".


Designs
Requirements
Estimated Impact of
Design
-> Requirements

US DoD. Persinscom **Impact EstimationTable:**

Design Ideas ->	*Technology Investment*	*Business Practices*	*People*	*Empowerment*	*Principles of IMA Management*	*Business Process Re-engineering*	*Sum Requirements*
	50%	[illegible]	5%	5%	5%	60%	185%
Availability 90% <-> 99.5% Up time	50%		5–10%	0%	0%	200%	265%
Usability 200 <-> 60 Requests by Users	[illegible]	[illegible]	5–10%	50%	0%	10%	130%
Responsiveness 70% <-> ECP's on time	50%	10%	90%	25%	5%	50%	180%
Productivity 3:1 Return on Investment	45%						303%
Morale 72 <-> 60 per month on Sick Leave	50%						251%
Data Integrity 88% <-> 97% Data Error %	42%						177%
Technology Adaptability 75% Adapt Technology	5%						160%
Requirement Adaptability ? <-> 2.6% Adapt to Change	80%						260%
Resource Adaptability 2.1M <-> ? Resource Change	10%	80%	5%	50%	50%	75%	270%
Cost Reduction FADS <-> 30% Total Funding	50%	40%	10%	40%	50%	50%	240%
Sum of Performance	*482%*	*280%*	*305%*	*390%*	*315%*	*649%*	
Money % of total budget	15%	4%	3%	4%	6%	4%	36%
Time % total work months/year	15%	15%	20%	10%	20%	18%	98%
Sum of Costs	*30*	*19*	*23*	*14*	*26*	*22*	
Performance to Cost Ratio	*16:1*	*14:7*	*13:3*	*27:9*	*12:1*	29.5 :1	

Figure 6.1.3 C. Source: An Agile Project Startup Week MASTER.
Also 111111 Unity Method of Decomposition into weekly increments of value delivery.
Case Study US Dept. of Defence. (10 min slides). *http://www.gilb.com/DL451*

6.1.4 EVO Backroom Frontroom

How to get a regular value stream, even when an architecture cannot be decomposed into small 2% value delivery cycles

I am a great believer in the potential to decompose almost any architecture suggestion into usefully-small value-delivery sub-architectures, as presented earlier in Part 4, Decomposition.

But I have to reluctantly admit that there are situations where this cannot be done, for example because there is a long delivery-time of a component (think military helicopters), or 'the new vaccine has not been scientifically approved yet'.

So, we long ago, found a useful solution (a major user was Philips Corporation of Holland [VP, and Figure 6.1.1 A]). I call it **Backroom/ Frontroom.**

Architectures which cannot immediately be decomposed into small, say 2%, increments, are placed in a Backroom, analogous to a Kitchen, until they are, in fact ready for deployment, as an increment ('Vaccine approved', for example).

In the meantime, in the Frontroom, the stakeholder-facing part of the organization, we deploy value-delivery increments, which are *ready* for deployment, and also deploy previous 'Backroom architecture' which has *become ready* for deployment.

There is in fact, usually more than enough, awaiting for small-cycle deployment, that we are not worried, just because something is 'brewing' in the Backroom. We have our hands full, for enjoying the value increases, and feedback, from the deployable increments.

The main point is, that in spite of some holdups, outside of our control, we can *invariably* deliver a value stream to the stakeholders, and 'feed the hungry lion' as I say.

Ultimately, all realistic architecture is released from the Backroom, for real implementation.

Let me summarize, or re-phrase this: you can always create an early (2nd week usually) frequent (weekly, 2%) real measurable value stream, even though *some* architecture elements, cannot be delivered initially.

I find it quite amazing how, for example, large government projects, like health and military systems [G: **Governeering:Government Systems Engineering Planning.** https://tinyurl.com/Governeering] totally fail to understand this simple idea. They drag on for years, at great expense, and fail scandalously.

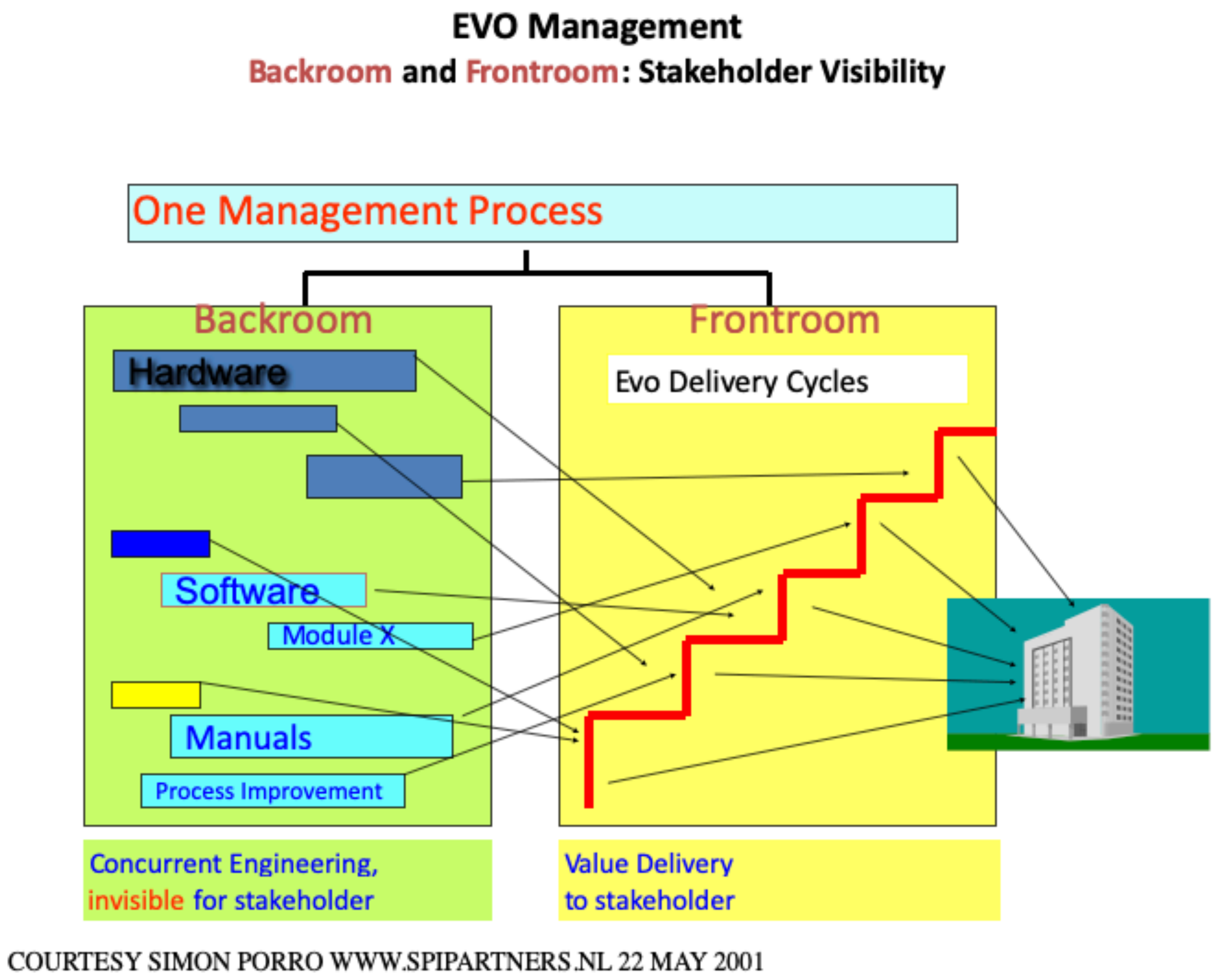
The politicians do not understand that they can demand evolutionary value delivery on all such systems. The technologists and suppliers are quite happy to waste time and money. They all seem to get paid well for failure, and never get sued or jailed, as they should be, for at least professional incompetence, and then for 'theft' of public funds (waste). Not to mention 'killing citizens', indirectly, for lack of better systems.

We are experiencing such a scandal, right now as I write (Akson E-Health project, Norway), and recently NAV (Social Security) system, same problems. They are 'coding agile', and have no concept of *incrementally delivering value*.

This makes me angry and sad as a citizen, and technologist. But of course there are even much worse nasty things governments do to their citizens on a daily basis, so 'don't complain'. Norway is a wonderful place to live, and with Oil money, has plenty to waste on large IT projects.

6.1.4 EVO Backroom Frontroom

A simple model of the backroom and frontroom relationships



EVO Management
Backroom and Frontroom: Stakeholder Visibility
One Management Process
Backroom
Hardware
Software
Module X
Manuals
Process Improvement
Frontroom
Evo Delivery Cycles
Concurrent Engineering, invisible for stakeholder
Value Delivery to stakeholder
COURTESY SIMON PORRO WWW.SPIPARTNERS.NL 22 MAY 2001

Figure 6.1.4 A. Source Gilb Slides Backroom Frontroom concepts subset

6.1.3 EVO Backroom Frontroom

Another simple model, from Kai Gilb

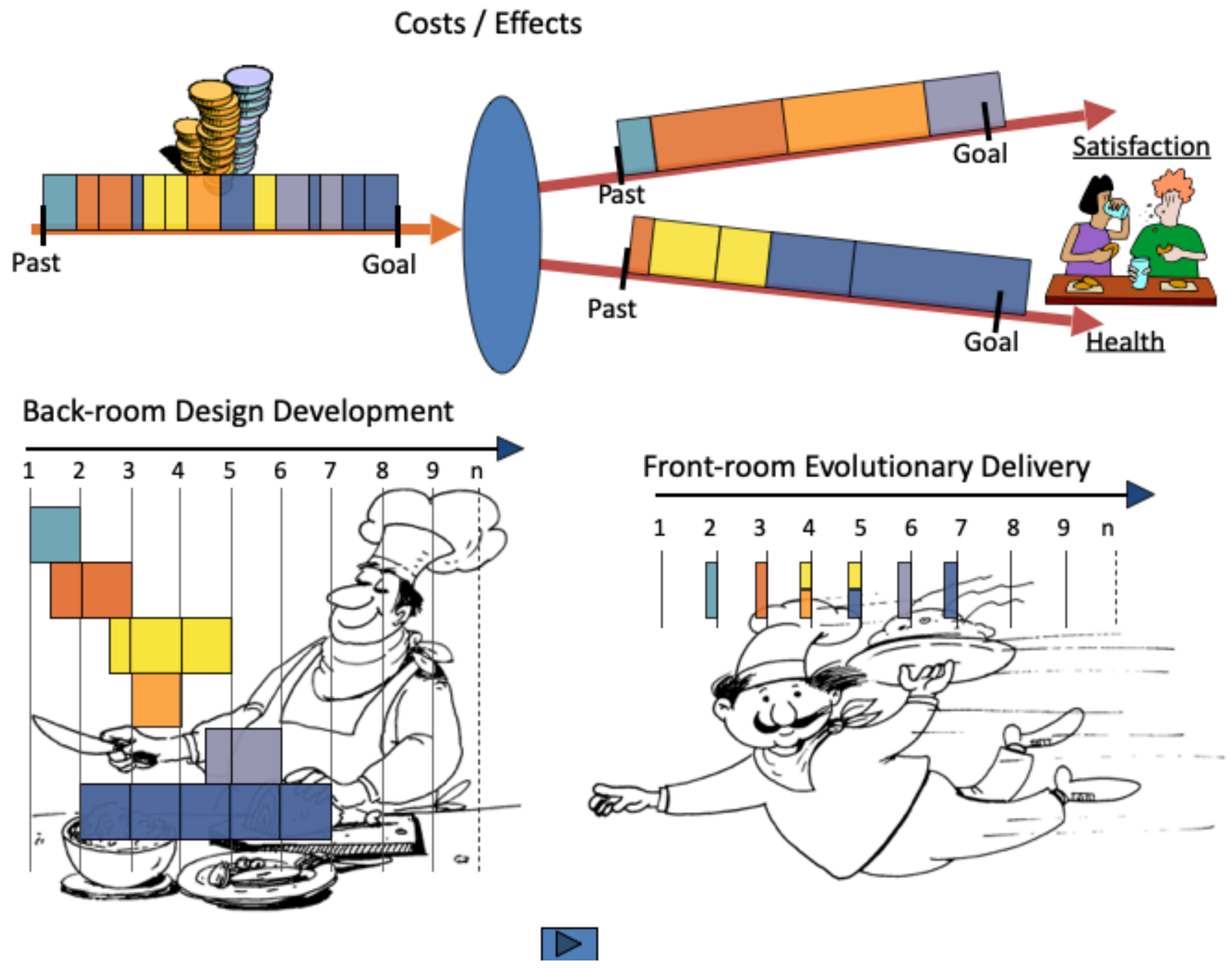
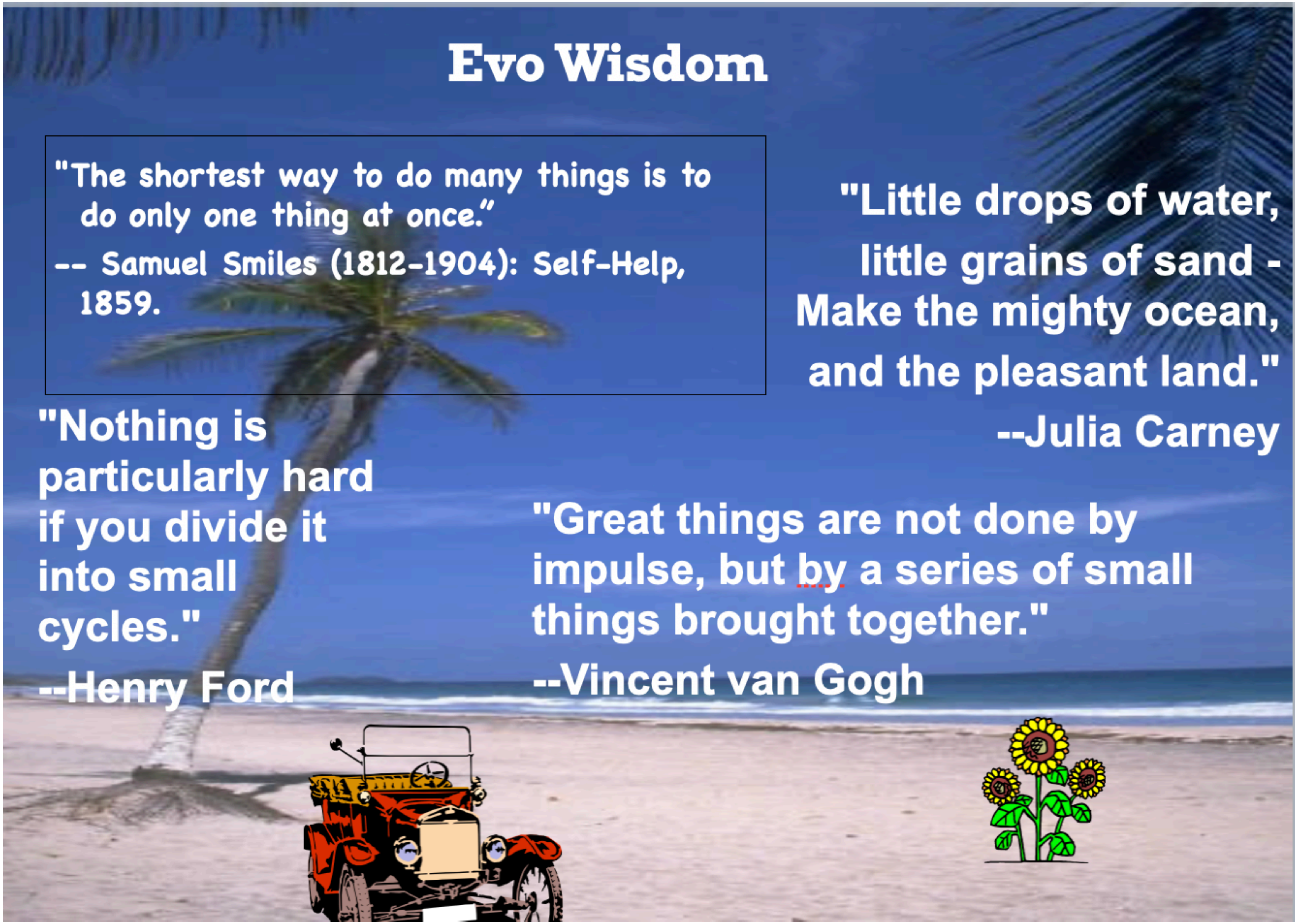


Figure 6.1.4 B. Source Gilb Slides Backroom Frontroom concepts subset



Evo Wisdom

"The shortest way to do many things is to do only one thing at once."

-- Samuel Smiles (1812-1904): Self-Help, 1859.

"Little drops of water, little grains of sand - Make the mighty ocean, and the pleasant land."

--Julia Carney

"Nothing is particularly hard if you divide it into small cycles."

--Henry Ford

"Great things are not done by impulse, but by a series of small things brought together."

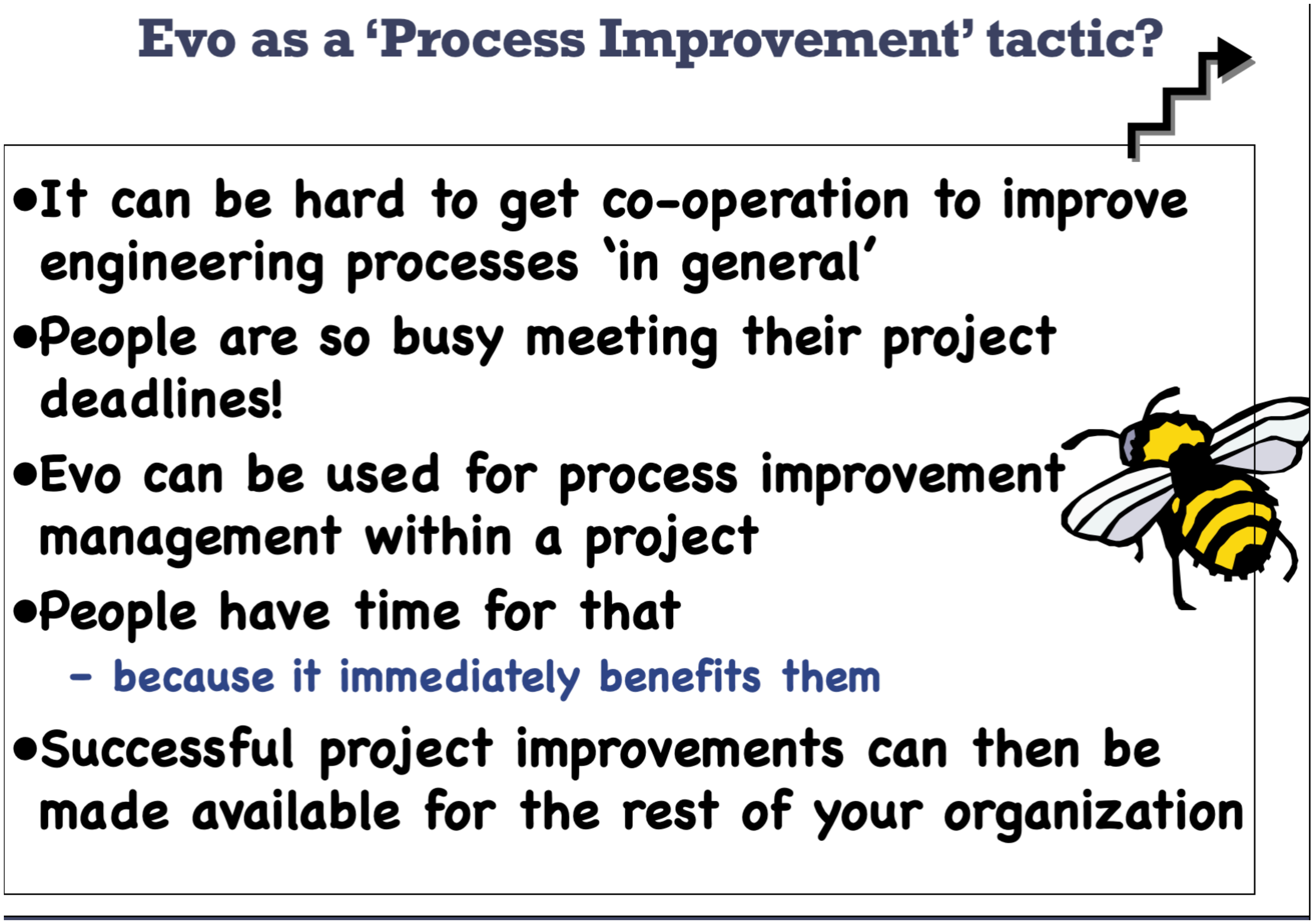
--Vincent van Gogh

Figure 6.1.4 C

Evo Attributes and Costs

- Management control of value
- Management control of costs
- Enforcing business thinking
  - Instead of technical thinking
- Flexibility for management to re-prioritize projects and expenditure
- Improves system maintenance culture
  - Because you 'maintain' at each step
  - Very low risk to do it and see if it works

The incremental nature of the Evo process, means that it is a good tool for organizational process improvement.



Evo as a 'Process Improvement' tactic?

- It can be hard to get co-operation to improve engineering processes 'in general'
- People are so busy meeting their project deadlines!
- Evo can be used for process improvement management within a project
- People have time for that
  - because it immediately benefits them
- Successful project improvements can then be made available for the rest of your organization

Figure 6.2

6.3 Architecture Value-Stream Delivery:

Evo Principles

1. The Principle of 'Capablanca's next move'

There is only one move that really counts, the next one.

2. The Principle of 'Do the juicy bits first'

Do whatever gives the biggest gains. Don't let the other stuff distract you!

3. The Principle of 'Better the devil you know'

Successful visionaries start from where they are, what they have and what their customers have.

4. The Principle of 'You eat an elephant one bite at a time'

System stakeholders need to digest new systems in small increments.

5. The Principle of 'Cause and Effect'

If you change in small stages, the causes of effects are clearer and easier to correct.

6. . The Principle of

'The early bird catches the worm'

Your stakeholders will be happier with an early long-term stream of their priority improvements, than years of promises, culminating in late disaster.

7. The Principle of 'Strike early, while the iron is still hot'

Install small steps quickly with stakeholders who are most interested and motivated.

8. The Principle of 'A bird in the hand is worth two in the bush'

Your next step should give the best result you can get now.

9. The Principle of 'No plan survives first contact with the enemy'

A little practical experience beats a lot of committee meetings.

10. The Principle of 'Adaptive Architecture'

Since you cannot be sure where or when you are going, your first priority is to equip yourself to go almost anywhere, anytime.

7 Da Vinci Principles: (Evo!)

<-Gelb, p.9

Curiosità

Insatiably curious, unrelenting quest for continuous learning

Dimostrazione

Commitment to test knowledge through experience, willingness to learn from mistakes. Learning for ones self, through practical experience

Sensazione

Continual refinement of senses. As means to enliven experience

Sfumato

Willingness to embrace ambiguity, paradox, uncertainty

Arte/Scienza

Balance science/art, logic & imagination, whole-brain thinking

Corporalità

Cultivation of grace, ambidexterity, fitness, poise

Connessione

Recognition & appreciation for interconnectedness of all things and phenomena, Systems thinking

Figure 6.3 A. 7 Da Vinci Principles: (Evo!) , Source: Michael Gelb, How To Think Like Leonardo Da Vinci

Figure 6.3 B. Tom at Da Vinci birthplace 2007

Da Vinci on Practical Feedback Principle

Leonardo, proudly described himself as:

- Uomo senza lettre (man without letters)
- Discepolo delle esperienza (disciple of experience)

"To me it seems that those sciences are in vain and full of error which are not born of experience, mother of all certainty, first hand experience which in its origins, or means, or end has passed through one of the five senses."

Figure 6.3 C. *Source: Gelb page 78*

Leonardo's persistence principle

"Although generally recognized as the greatest genius of all time, Leonardo made many colossal mistakes and staggering blunders."

"Despite mistakes, disasters, failures, and disappointments, Leonardo never stopped learning, exploring, and experimenting.

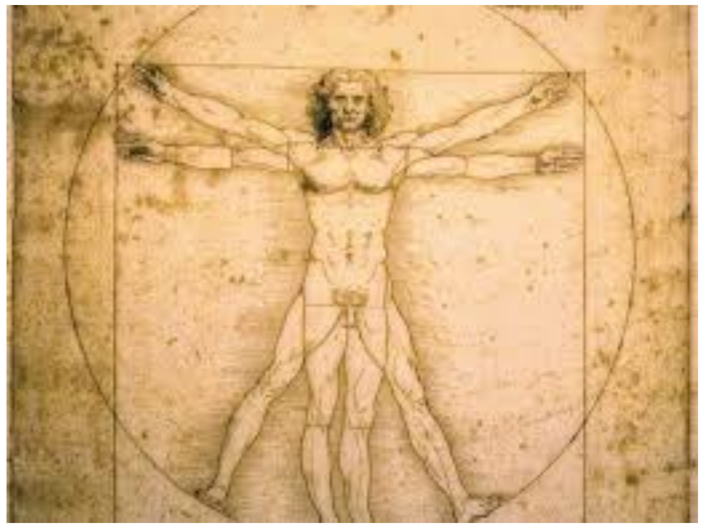
He demonstrated Herculean persistence in his quest for knowledge."

Leonardo wrote:

"I do not depart from my furrow.

"Obstacles do not bend me"

"Every obstacle is destroyed through rigor"



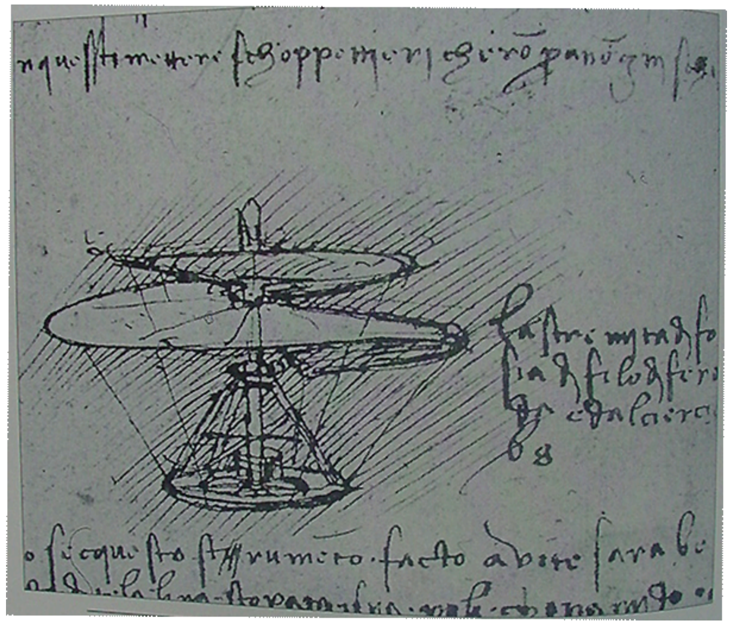


Figure 6.3. D
Source: Gelb

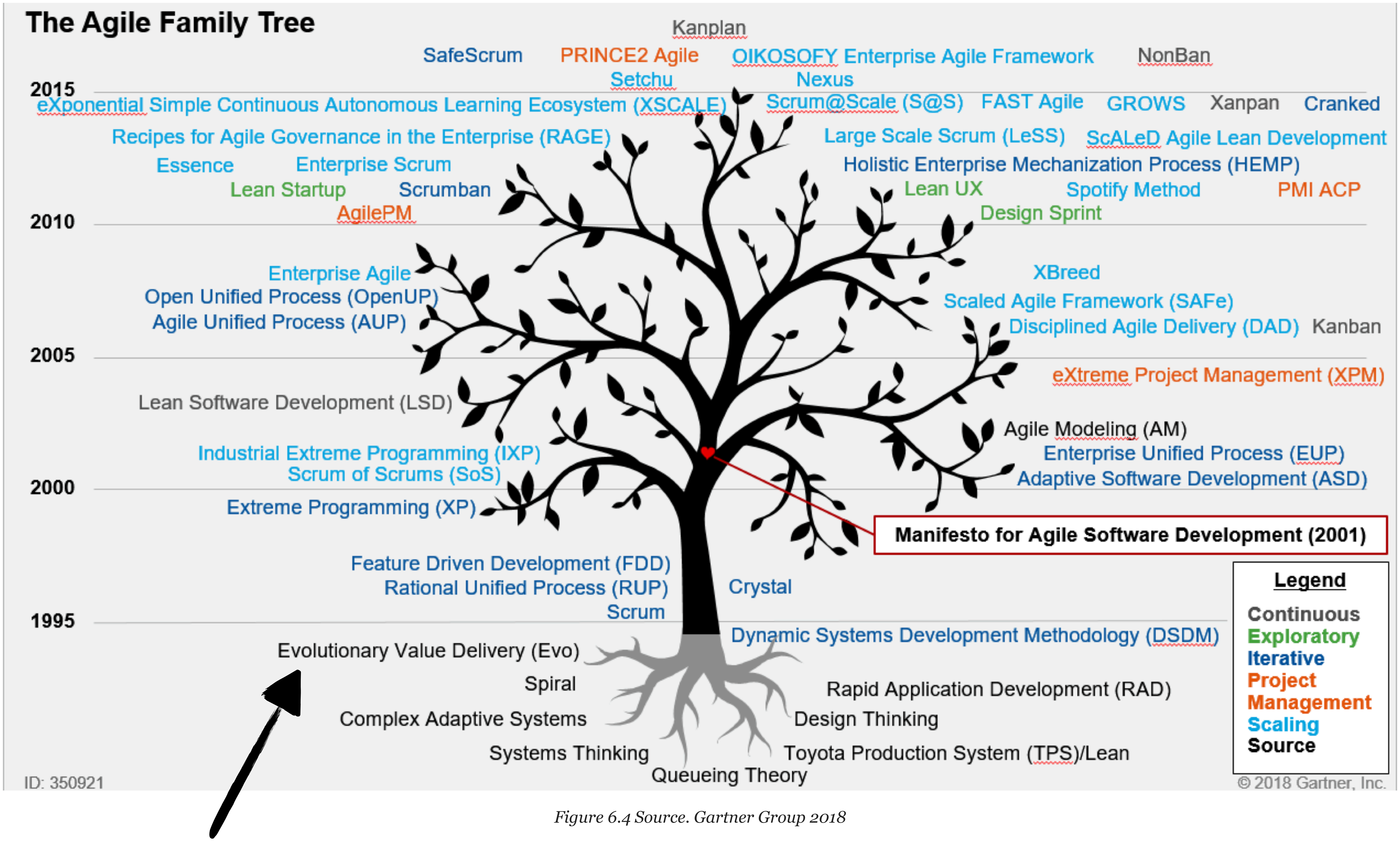

The Agile Family Tree
Kanplan
SafeScrum
PRINCE2 Agile
OIKOSOFY Enterprise Agile Framework
NonBan
Setchu
Nexus
2015
eXponential Simple Continuous Autonomous Learning Ecosystem (XSCALE)
Scrum@Scale (S@S)
FAST Agile
GROWS
Xanpan
Cranked
Recipes for Agile Governance in the Enterprise (RAGE)
Large Scale Scrum (LeSS)
ScALeD Agile Lean Development
Essence
Enterprise Scrum
Holistic Enterprise Mechanization Process (HEMP)
Lean Startup
Scrumban
Lean UX
Spotify Method
PMI ACP
AgilePM
Design Sprint
2010
Enterprise Agile
XBreed
Open Unified Process (OpenUP)
Scaled Agile Framework (SAFe)
Agile Unified Process (AUP)
Disciplined Agile Delivery (DAD)
Kanban
2005
eXtreme Project Management (XPM)
Lean Software Development (LSD)
Agile Modeling (AM)
Industrial Extreme Programming (IXP)
Enterprise Unified Process (EUP)
Scrum of Scrums (SoS)
Adaptive Software Development (ASD)
2000
Extreme Programming (XP)
Manifesto for Agile Software Development (2001)
Feature Driven Development (FDD)
Rational Unified Process (RUP)
Crystal
Scrum
1995
Dynamic Systems Development Methodology (DSDM)
Evolutionary Value Delivery (Evo)
Spiral
Rapid Application Development (RAD)
Complex Adaptive Systems
Design Thinking
Systems Thinking
Toyota Production System (TPS)/Lean
Queueing Theory
Legend
Continuous
Exploratory
Iterative
Project Management
Scaling
Source
ID: 350921
© 2018 Gartner, Inc.

Figure 6.4 Source. Gartner Group 2018

6.5 Architecture Value-Stream Delivery: Evo not needed when…

When You Do Not Need Evo

You do not need Evo if

1. There is no instability of requirements
2. There is no pressure on resources, to meet requirements
3. There is no volatility (frequent change) on the cost-or-ability of technology
4. There is no 'corruption', under pressure, to carry out planned 'architecture'
5. There is no need for early deliveries
6. Lateness of everything , by factor 3.14, is tolerable
7. Nobody is 'green',

   (everybody knows all they need to know about the complex new advanced state-of-the-art system they are building: nothing to learn)

Risk and Enterprise Architecture

The ISO Risk standard [**P15:** ISO 31000:2018] defines risk as

risk

“effect of uncertainty on objectives”

This same standard is used in academic papers discussing EA and Risk [R16]

The Committee of Sponsoring Organizations of the Treadway Commission (COSO) view of ERM (Enterprise Risk Management) is

that

"Every entity exists to provide value for its stakeholders”

So, with these key words, ‘***effect, uncertainty, objectives’, and then ‘value, stakeholders’.***

You will recognize that these concepts are central to this book and my SEA ideas.

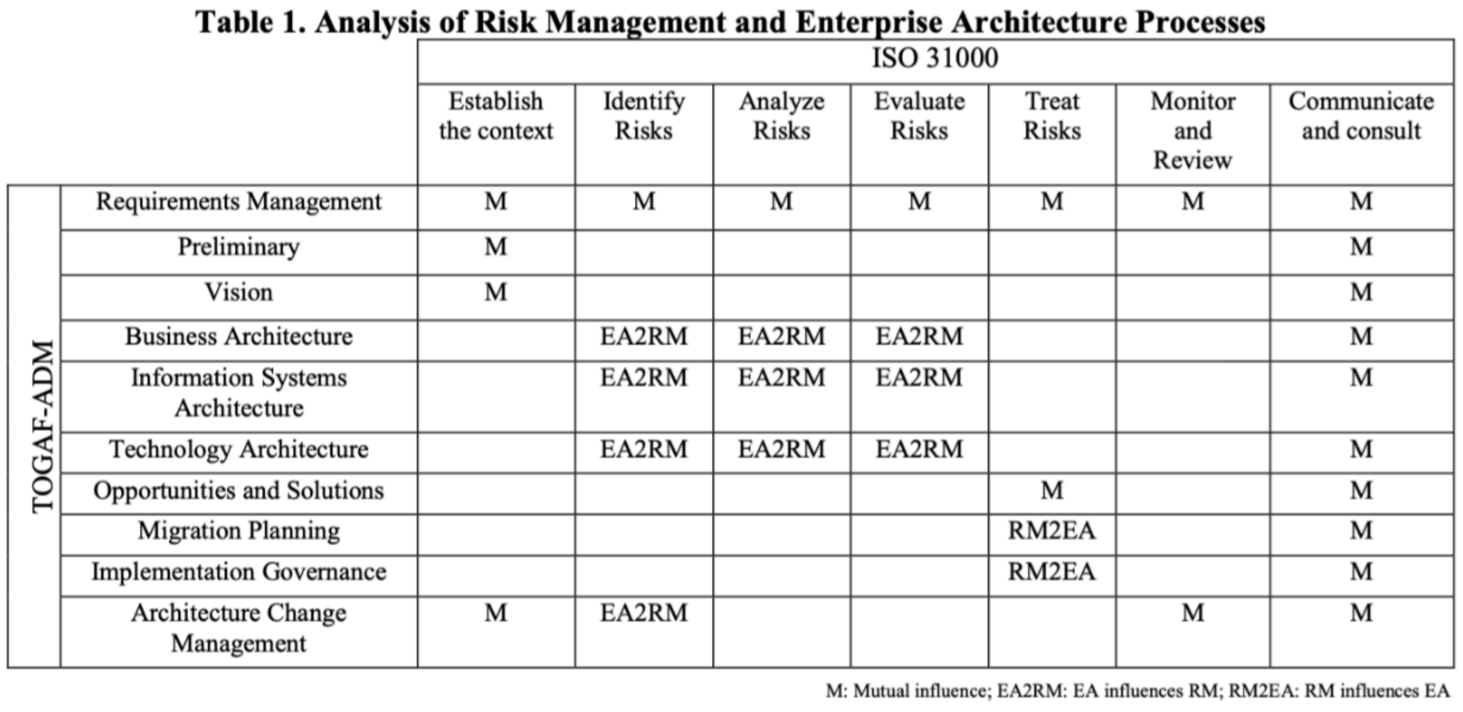


Table 1. Analysis of Risk Management and Enterprise Architecture Processes

		ISO 31000						
		Establish the context	Identify Risks	Analyze Risks	Evaluate Risks	Treat Risks	Monitor and Review	Communicate and consult
TOGAF-ADM	Requirements Management	M	M	M	M	M	M	M
	Preliminary	M						M
	Vision	M						M
	Business Architecture		EA2RM	EA2RM	EA2RM			M
	Information Systems Architecture		EA2RM	EA2RM	EA2RM			M
	Technology Architecture		EA2RM	EA2RM	EA2RM			M
	Opportunities and Solutions					M		M
	Migration Planning					RM2EA		M
	Implementation Governance					RM2EA		M
	Architecture Change Management	M	EA2RM				M	M

M: Mutual influence; EA2RM: EA influences RM; RM2EA: RM influences EA

Figure 7.0. Source [R16]

I am going to use little time, complaining about the conventional EA [VR, where I make specific complaints], and its real detailed approach to risk. I find it ‘underwhelming’. It starts and ends with the **total lack of quantification, and clarity, for ‘stakeholder value objectives’**. ‘Game Over’. I will assume the reader sees my point, and sees in detail, in this book, what I mean by **‘quantifying stakeholder value objectives’**.

So I will concentrate on how the System Enterprise Architecture (SEA) can help us manage risks, better than what I have seen in the EA literature.

Should any reader care to enlighten me about equally good or better risk practices in EA, I would be happy to learn. But my clients and students have not done so yet, in spite of decades of provocation.

If you do not know some EA Architecture practices better or equal, would you consider adopting SEA ? It’s free, or do you prefer to pay for mediocracy?

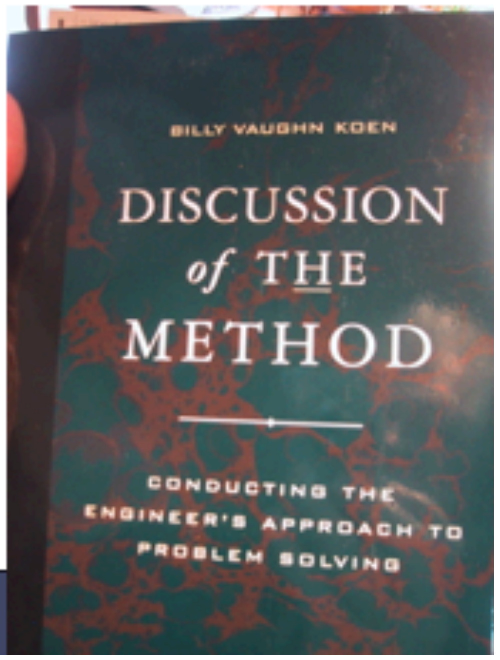
Koen on Risk Control

Make small changes in the sota:

'Sota' = Engineering State Of The Art Heuristics <-Koen, Discussion, p. 48

Always give yourself a chance to retreat; and

Use feedback to stabilize the design process



www.Gilb.com

EA Risk Principles

1. **GOAL THREATS**: All uncertainty in enterprise architecture poses a threat to the planned objectives outcome, or delivery deadline.

2. **LEARN FAST**: The best way to deal with risk is to, early, frequently, and measurably deliver a value stream: learn fast how architecture actually works, and correct the architecture.

3. **EA RISK**: The total **net risk** of damage to enterprise stakeholder valued results, is a function of the degree and type of threats, the threat mitigation (avoidance, detection, thwarting), the degree of actual attacks, attack mitigation, and the damage mitigation.

4. **EA MITIGATION**: the Systems.Security Enterprise Architect cannot do much but to analyze and observe the initial threat stream, type, frequency and potentially emerging threat; but based on this, they can architect comprehensive cost-effective mitigation strategies.

5. **WHOLE SYSTEM**: The EA architect, in analyzing and designing damage mitigation must absolutely take a full systems perspective: including world politics, world economy, world health, national political and economy, psychology of stakeholders, antagonistic stakeholders, long term costs not just to the enterprise but to all stakeholders, ethics, corporate policy, sustainability, and more. IT Geeks need not apply.

6. **LIMITS TO MITIGATION:** threats are unlimited, unknown, unpredictable, as are their damage consequences. You cannot possibly deal perfectly with them all. So, you are going to have to do a cost-benefit prioritization of the sequence you will invest in damage reduction.

7. **ENGINEERING RISK OUT**: You will need to systems engineer it. But you can deal with common and dangerous threats. You can invest in cost-effective mitigation. You can be responsible and transparent about the Risk Management process.

8. **RESPONSIBILITY**: The Board Level, in particular the CEO, is responsible for making Risk Architecture happen, and charging the CTO, CIO and other C-level executives with visibly doing the Architecture, making the investments and being very transparent about what they are doing, and not, and why. Empower the Architect.

9. **PREVENTION**: upstream prevention mitigation is the most cost-effective line of defence.

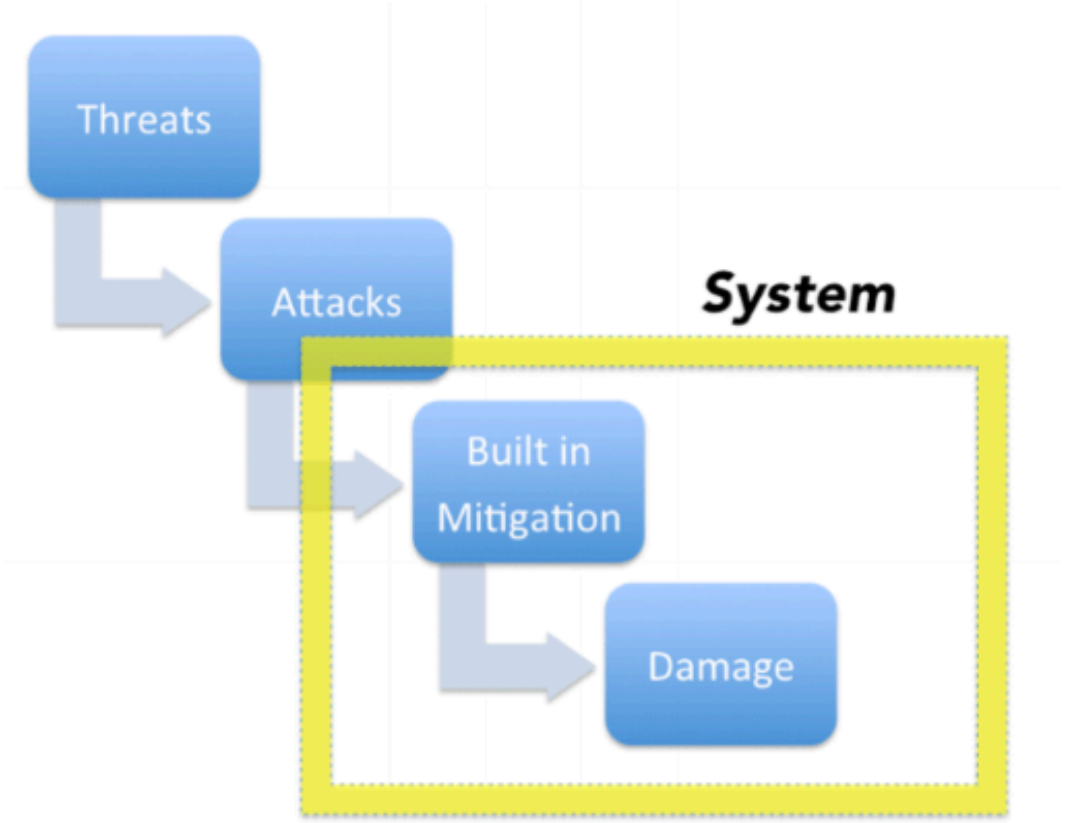
10. **SECURITY**: is intimately integrated with Risk.

7.1. Architecture Risk Management: EA Risk Management Principles

Risk of unmitigated damage to any enterprise and related stakeholder values , is very much up to the security and risk architecture
There are many areas of mitigation, designed by the architect, which can prevent, reduce, capture and compensate for threats to the Enterprise
The CTO and CIO question is: <u>have you invested in doing the architecture for mitigation properly?</u>
It is non trivial

1. **You have to set quantified objectives and risk mitigation budgets.**
2. **You have to 'self-insure' for some risks and damage**
3. **You have to prioritize building the mitigation architecture, in value/costs sequence.**

Page 9 of 70 © tom@Gilb.com **Chapter 7. Risk Management**



Threats: A 'threat' is something that can *potentially* cause some degree of project failure, lack of success or negative consequences. It is distinguished from an Attack, which is a *successful* penetration of the Threat into a system. The Threat has 'materialized' in practice. It is distinguished from a Risk, which is a result of the combined effect of a Threat/Attack and the corresponding defenses (Mitigation)

Figure 7.2 The risk environment. Threats may turn into real attacks, which get confronted with our planned mitigation. If the mitigation plan is unsuccessful, damage results, of various kinds. Notice that 'built-in to the system mitigation' is one type of mitigation. We can also mitigate risks at earlier stages, such as in planning, contracting, building trials, tests etc.

Figure 7.1 A. Source [P18, VP Chapter 7]

7.1. Architecture Risk Management: EA Risk Management Principles, Stakeholders

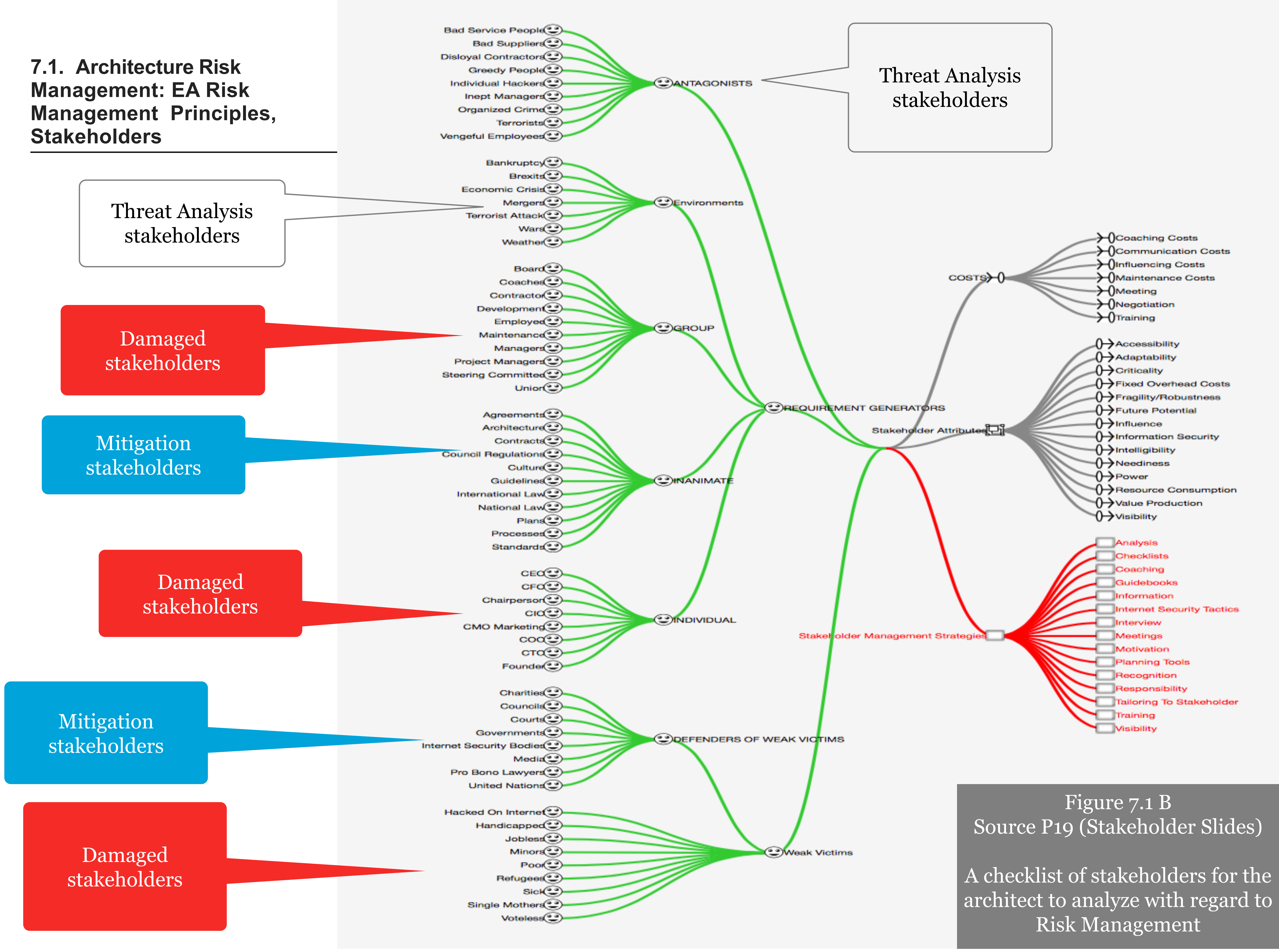
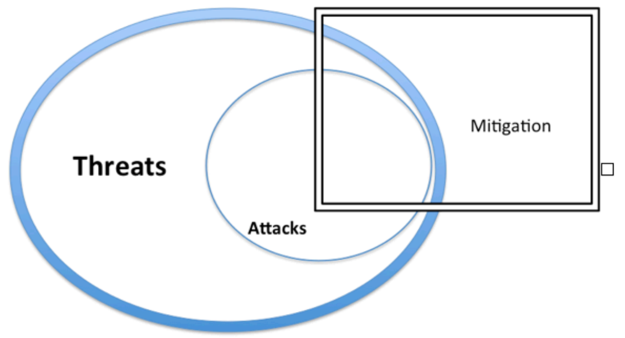


Figure 7.1 B
Source P19 (Stakeholder Slides)

A checklist of stakeholders for the architect to analyze with regard to Risk Management

Risk Methods - totally pervasive in SEAL



The methods in this book are **absolutely all 'risk management' methods**. They've are all directed at making sure the *enterprise objectives*, at all levels, from top down, are delivered. That this the official definition of risk, above. Delivering the objectives correctly.

In several of my Referenced books, like *Value Planning, 100 Practical PlanningPrinciples, Technoscopes and Competitive Engineering* you will find 10 Chapters sub-divided into 10 sub-chapters. In other words *100-components each*. Most of those 100 components consist of several (5 to 10) sub-tools. I promise you that there is an argument that every one of those 100 components will *help you deal with ris*k, and in particular with the defined *risk for Enterprises* of avoiding 'deviating from their critical stakeholder value objectives'.

The all-is-risk argument is made in my writings directly, or should be fairly obvious to the intelligent analyst. I have never heard any of my students or readers, all very intelligent, challenge me on that point, and I have been making it for over 30 years. You can try, but make an interesting bet with me, and be prepared to lose.

So a great deal of the detailed ideas, are *right here in this book*. The stakeholder points I just made above, the objectives with scales and points on the scale partly discussed above, and in much greater detail below. Not to mention in the extensive, mostly free, References. Don't waste our energy denying what I am claiming. Get going and get some experience in the 100 Tools, and *then* make up your own mind.

Having dispensed with the details, by *claiming they are all other written pages, case studies, and experiences here and in References*; let me try to give an overview, right here.

7.2.2 Architecture Risk Management: specific risk methods areas.

Risk Methods - the big 4

The SEAL (Planguage) Main Tools for Enterprise Risk Management.

1. **STAKEHOLDERS**: The rich stakeholder-analysis tools, far beyond wide-spread 'User' and 'Customer ideas'. (Figure 7.1B). Each *missed stakeholder* is a big risk. These stakeholders are all directly aligned to one-or-more quantified values, which can be adopted as enterprise objectives.

2. **QUANTIFIED VALUES**: The rich structures and quantification of stakeholder-value objectives (SEA 10. See this for more detail). And see the **value *analysis*** methods ('12?', 'Planalysis', see References) that come with SEA Language. These '*stronger than most every other' methods* are designed to make sure the Enterprise knows exactly what their objectives *really* are, and the Enterprise Architect has extremely-clear architecture targets.

3**. IET**: The Impact Estimation Table gives direct quantified connections, estimated and measured, between **enterprise objectives** and **enterprise architecture**. Goodbye EA fuzzy blah blah [VR].

4. **Evo**: Evolutionary stakeholder value delivery is the ultimate real-time guarantee that we will deliver Enterprise Objectives, largely without fail, and experience says, exceeding expectations. The first 3 tools (Stakeholders, Quantified Objectives, Impact Estimation) lay the groundwork for this. They make pretty sure that the architecture matches the multiple objectives, and resource constraints. But then the 2% value delivery steps, guarantee that we cannot get big problems and deviations, before we get credible feedback, and an architects opportunity to adjust the architecture in the right enterprise target and cost directions.

"Risk comes from not knowing what you're doing"

– Warren Buffett (1930-)

Value for costs is more or less guaranteed: No Risk.

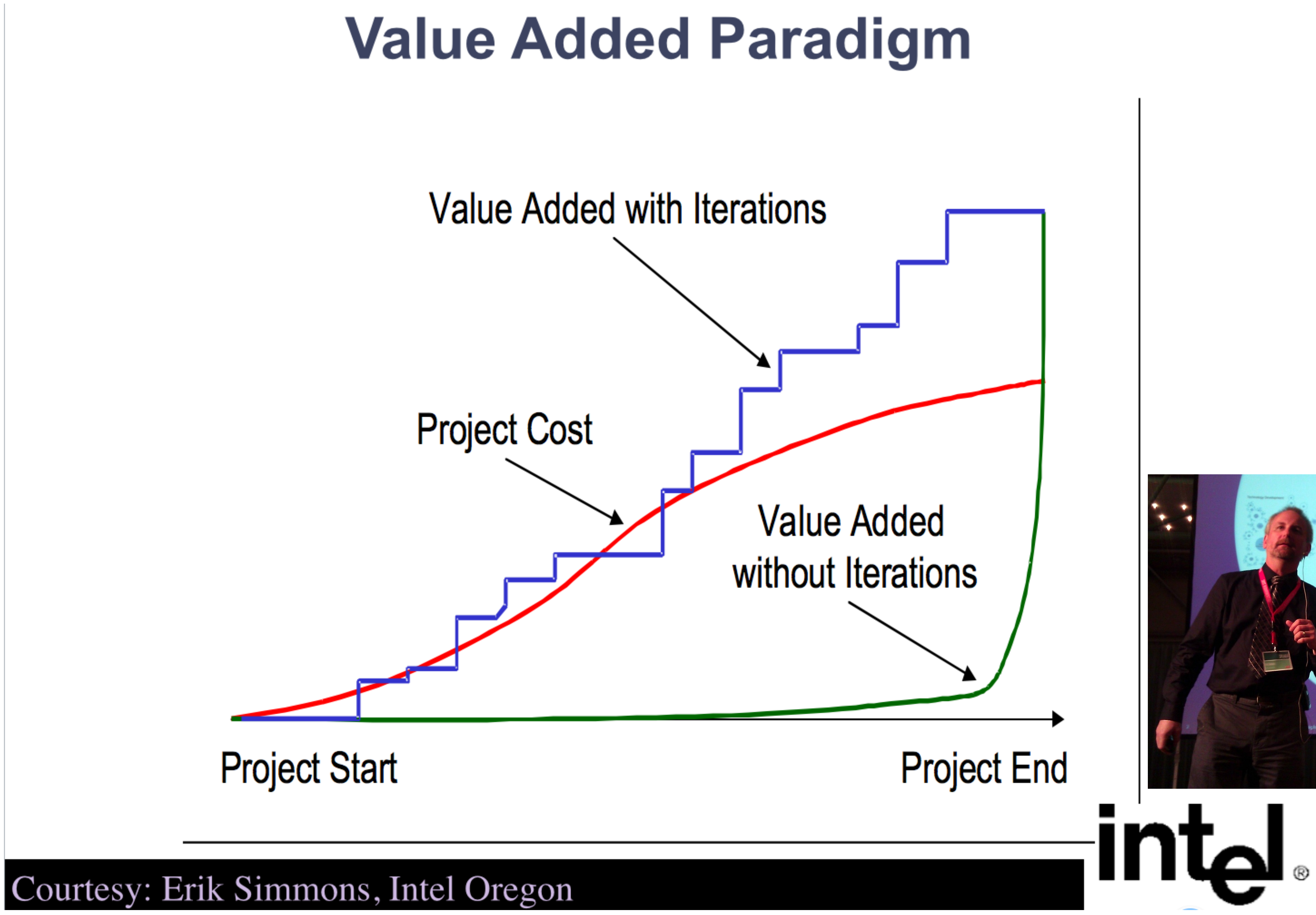


*Figure 7.2.2. Source: Value Planning [VP] Figure 7.4 A. One aspect of dealing with value delivery early and frequently is that the value is **in place much earlier**. The main point however is that you can be sure you **really did get value** from your strategy ideas. You do not risk that they totally fail you, as so many projects actually do. I like Erik's remark in his Foreword to my 'Competitive Engineering' [1] book: 'This stuff works!'. Erik Simmons and his staff have trained over 20,000 Intel Engineers to use Planguage in their daily work. They volunteer to learn it, it is not a required enterprise 'standard'.*

7.2.3. Architecture Risk Management: Space and Military Experience

No Risk. No damage to objectives. Profit. Reputation.

Let me introduce Harlan Mills. The 'Leonardo da Vinci' of Software Engineering. He was given a very difficult 'Enterprise' problem to solve, by his IBM Federal Systems Division management.

> ***" Every time we win a government contract, as lowest bidder, with fixed prices, fixed high quality and fixed deadlines, we lose money. Can you fix that? "***

His '*Cleanroom'* team cracked the problem, over a ten year period. Proving on real projects, that they could deliver the most-advanced technology, military and space, 'on time and under budget' – every time, years in a row. **This is not a'project success, alone. It is an Enterprise success!**

Not bad. In fact, 'perfect project management'!

Why can't we all do as well? We can if we follow their recipe! The key ideas are the same as this book is preaching (Same as Evo). His world was software, but the principles we are offering for consideration here are *universal*.

IBM was not in the marketplace 'selling' this 'Cleanroom' method (as they do 'Watson', 'Rational' and other method products, which are more widely adopted).

IBM was not in the 'methods' market at the time. So most managers never heard of it, and its remarkable results. Sometimes the best things in life *are* free!

Let Mills speak for himself *[IBM Systems Journal 4/1980, P3]*

"Software Engineering began to emerge in FSD" *(IBM Federal Systems Division, later a part of Lockheed Martin Marietta, and other mergers later)* ***) "some ten years ago, in a continuing evolution that is still underway:***

Ten years ago general management expected the worst from software projects – cost overruns, late deliveries, unreliable and incomplete software

Today management has learned to <u>expect</u> on-time, within budget, deliveries of high-quality software.

A Navy helicopter ship system, called LAMPS, provides a recent example. LAMPS software was a four-year project of over 200 person-years of effort, developing over three million, and integrating over seven million words of program and data for eight different processors distributed between a helicopter and a ship in 45 incremental deliveries." *(Note, that is about 2% of time to deadline for each delivery step).*

"Every one of those deliveries was on time and under budget.

A more extended example can be found in the NASA space program" *(Space Shuttle Ground Software)."*

- "Where in the past ten years, FSD has managed some 7,000 person-years of software development, developing and integrating over a hundred million bytes of program and data for ground and space processors in over a dozen projects.

- There were few late or overrun deliveries in that decade, and none at all in the past four years."

Wow!

Perfect Large-scale long-term high-tech project management. Are *you* that good? Go back to the failure rates, persisting today, cited in 0.0 above.

It is worth noting, that everybody else in the FSD Enterprise Domain, space and military at that time, was doing 'big bang' (aka Waterfall) projects, and failing; at taxpayer expense, as IBM also did previously. MIlls is using 'agile as it should be', with quantified values, like for 'availability' driving them. [S6. **IBM FSD MIlls and Quinnan Slides**. <u>http://concepts.gilb.com/dl896</u> (see also P3.1 to P3.3)]

8.0 Architecture Enterprise Alignment: Basic ideas

The Enterprise Architecture (EA) is expected to

1. 'be aligned' (itself, EA), at all times with changing Enterprise Objectives, stakeholders, constraints, and strategies.
2. EA should help the Enterprise become *internally aligned* with internal management Objectives, stakeholders, constraints, and strategies. At *all* levels and sections of the enterprise.
3. Aligned: means
   1. **not** in unnecessary and destructive conflict with each other,
   2. and **is** supporting synergy with each other.

Alignment of Values and Visions

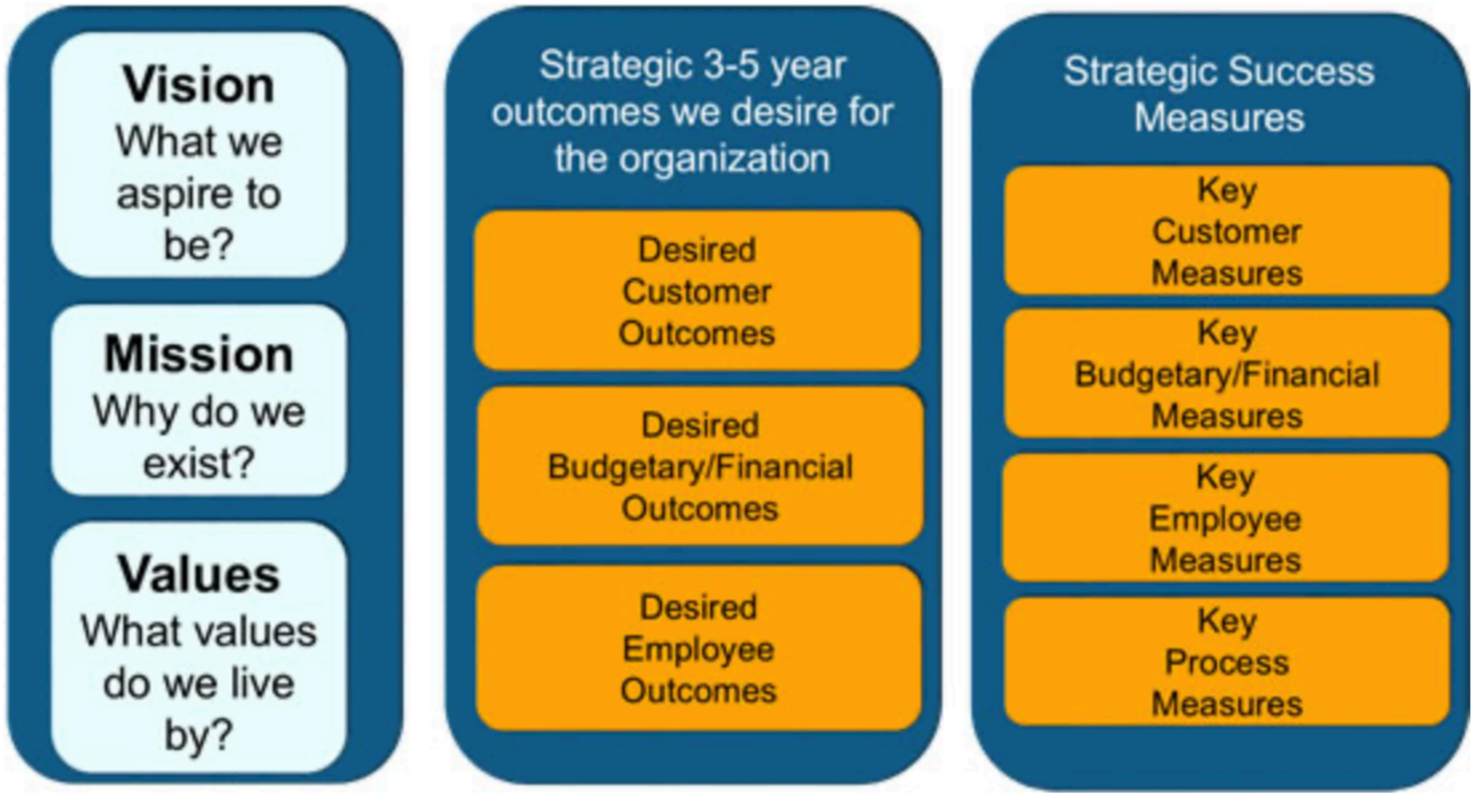

Vision
What we aspire to be?
Mission
Why do we exist?
Values
What values do we live by?
Strategic 3-5 year outcomes we desire for the organization
Desired Customer Outcomes
Desired Budgetary/Financial Outcomes
Desired Employee Outcomes
Strategic Success Measures
Key Customer Measures
Key Budgetary/Financial Measures
Key Employee Measures
Key Process Measures

Figure 8.0
Source https://integrispa.com/blog/the-four-dimensions-of-lean-culture-enterprise-alignment/
For Vision people see

[B3:VE.**Vision Engineering].** .(free download, 60 pages)

Aligned: Enterprise Architecture is synchronized with all significant external and internal Enterprise forces, and plans: updated, precise, supporting, relevant, non-conflicting, transparent, and future oriented.

Source TSG draft definition 100920

Rules and Policies of

Enterprise Architecture Alignment.

1. PRECISE OBJECTIVES: All objectives, visions, and value requirements must be unambiguously clear, structured and quantified; so that correct alignment is possible at all.

2. DIGITAL OBJECTIVES: All objectives, will be digitally intelligible, so that automatic alignment, and precise alignment is technically possible, in large dispersed enterprises.

3. ALL KNOWN RELATIONS: All objectives, in their specification object, will contain all necessary specification about relationships to everything currently known, like stakeholders, strategies, and responsibilities. Links will be digital.

4. ALIGNMENT RESPONSIBILITY: The specific 'position or organizational unit' which is responsible for making alignment, keeping alignment, and quality controlling alignment; will be explicitly named in relevant specification objects, with a digital link to them, and to their specific updating responsibilities, for specific sets of specifications.

5. TABLES SHOW ALIGNMENT DEGREE: The degree of alignment of any supporting 'means' to our 'ends' (strategies, architecture), or 'means objectives', to any next level, or scattered related objectives, will be accounted for on an Impact Estimation Table for clarity.

Vision
What we aspire to be?
Mission
Why do we exist?
Values
What values do we live by?

6. MULTIDIMENSIONAL ANALYSIS: The alignment will never be a narrow, one-to-one relationship alone, but will always be multiple-objectives and costs; impacted by multiple-supporting means-objectives or strategies. Side-effects will be clearly accounted for.

7. THEORETICAL AND REAL ALIGNMENT: There will be two major temporal concepts of IE Table alignment: 1. Planned and Estimated Alignment, and 2. Actual current alignment, as measured.

8. INTERNAL AND EXTERNAL ALIGNMENT: There will be two major scope concepts of alignment: (1) Internal in the enterprise, and (2) from Enterprise to the external world of stakeholders, and all forces we must contend with. May the force....

9. ALIGNMENT EXECUTIVES: The CEO is ultimately responsible for the mechanisms and their quality, for alignment. Supported by all C level executives. F: Financial, M: Marketing, I:IT, etc.

10. ALIGNMENT EFFICIENCY: The strategies to track alignment, and promote alignment, will always have a clearly enumerated value-for-costs, or ROI. The whole purpose is meeting Enterprise objectives better

12.3.1.15 Estimating Future Levels of Value Quantities. IET

OK we can quantify our ‘soft’ values. We can then quantify our designs’ impact on those values. What can we do with such tools? They are very general tools, like the Swiss Army knife.

They can be used for any systems, in any domain, large or small. Even the largest [S13, P14]. As to the smallest, they can run on intuition, and do little damage.

The range of applications is without any known limit, according to my experience. I can hardly think about any challenging new problem,(and I seem to find them weekly) before I must bring out these tools to think about it.

I suppose Planguage is a tool similar to language and mathematics, a systems engineering language, a planning language.

The origin of Planguage was about 1960-1 when I worked at an insurance company (Storebrand, Oslo) which wanted my help selecting their first electronic computer. I realized that there was far more to acquiring a computer than its speed, which was the dominant talking point then. There were things like the degree of long-term service, and maintenance, that could be expected, and whether the supplier would at all be in Business in 5-years time. So I realized that the evaluation of computers, would be the evaluation of many soft values, not all about the hardware. I also knew my client, the Actuaries were both interested in numbers, and interested in the long term. So I drafted something looking very much like an Impact Estimation table, minus many of the developments I have shared with you in this book. I was not yet an expert on turning values into numeric scales. But I treated it as a multi-value multi-cost problem. We made a good choice (IBM 1401) and IBM did indeed stay in business to this day. None the others did. That was 60 years ago. And I used the time since, to improve the methods. And to try them out in a wide variety of applications. It made life quite amusing.

So, this first application of multi-dimensional evaluation was for ***comparing alternative options.***

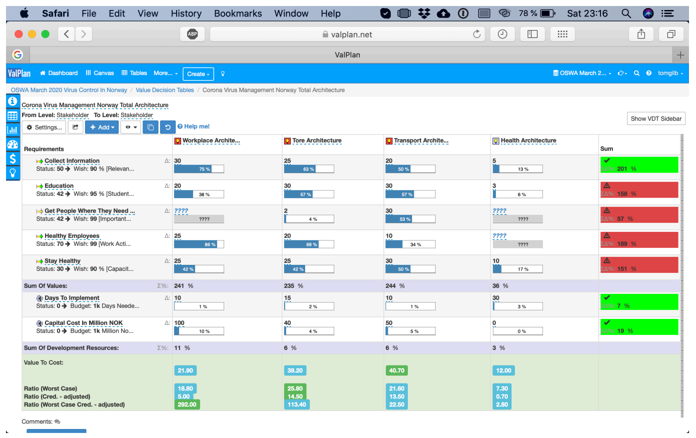
Other applications followed, a list on the next page. Here is a list of the basic applications.

1. Comparing complex options.

2. Building and evaluating a total architecture.

3. Managing a project.

4. Presenting complex systems, selling them.


Safari File Edit View History Bookmarks Window Help
valplan.net
ValPlan
Dashboard Canvas Tables More... Create
OSWA March 2... tomgilb
Settings... Add Help me!
Show VDT Sidebar

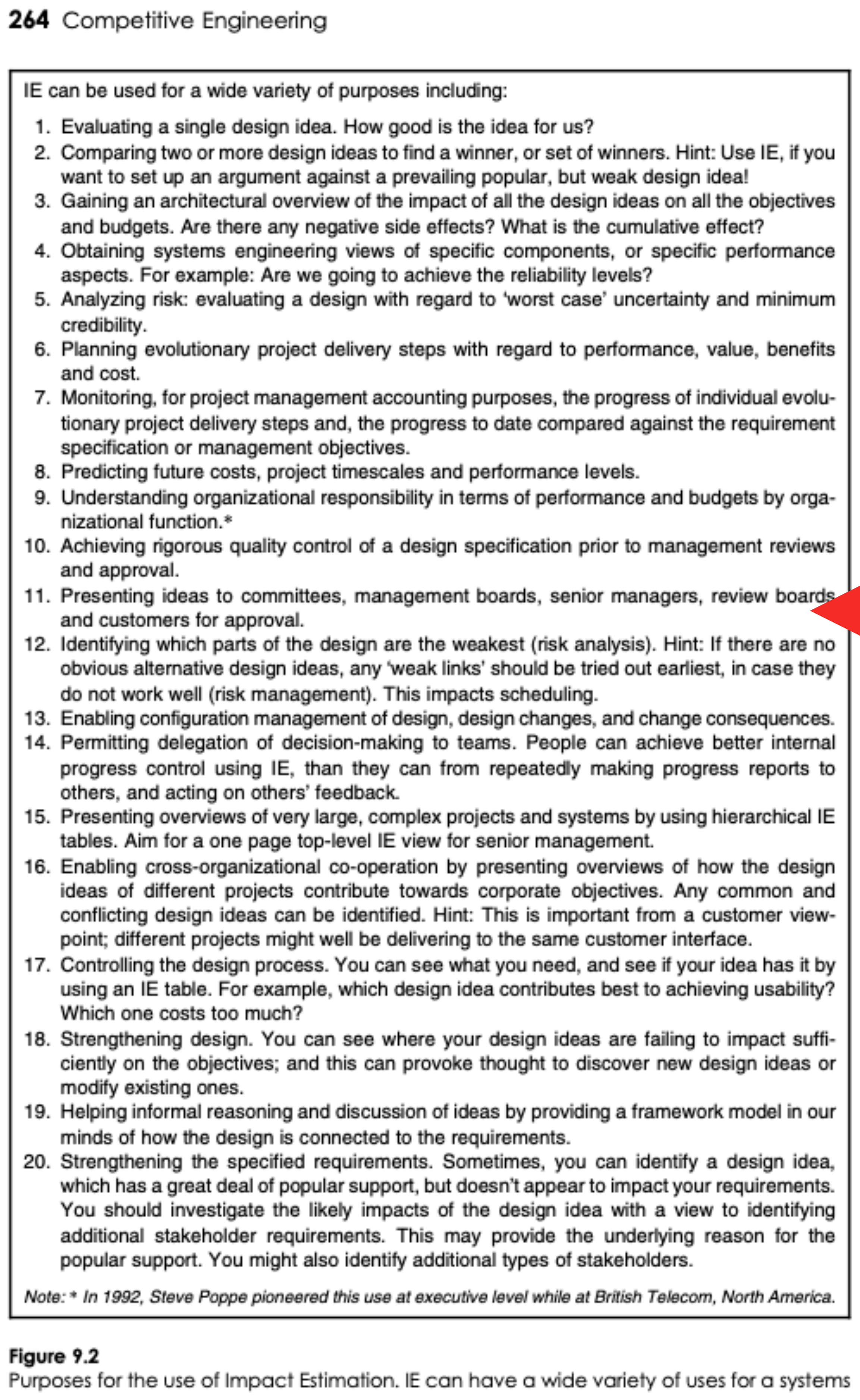
OSWA March 2020 Virus Control In Norway / Value Decision Tables / Corona Virus Management Norway Total Architecture

Corona Virus Management Norway Total Architecture
From Level: Stakeholder **To Level:** Stakeholder

Requirements		Workplace Archite...	Tore Architecture	Transport Archite...	Health Architecture	Sum
Collect Information Status: 50 → Wish: 90 % [Relevan...	Δ:	30 75 %	25 63 %	20 50 %	5 13 %	✓ ΣΔ%: 201 %
Education Status: 42 → Wish: 95 % [Student...	Δ:	20 38 %	30 57 %	30 57 %	3 6 %	⚠ ΣΔ%: 158 %
Get People Where They Need ... Status: 42 → Wish: 99 [Important...	Δ:	???? ????	2 4 %	30 53 %	???? ????	⚠ ΣΔ%: 57 %
Healthy Employees Status: 70 → Wish: 99 [Work Acti...	Δ:	25 86 %	20 69 %	10 34 %	???? ????	⚠ ΣΔ%: 189 %
Stay Healthy Status: 30 → Wish: 90 % [Capacit...	Δ:	25 42 %	25 42 %	30 50 %	10 17 %	⚠ ΣΔ%: 151 %
Sum Of Values:	Σ%:	241 %	235 %	244 %	36 %	
Days To Implement Status: 0 → Budget: 1k Days Neede...	Δ:	10 1 %	15 2 %	10 1 %	30 3 %	✓ ΣΔ%: 7 %
Capital Cost In Million NOK Status: 0 → Budget: 1k Million No...	Δ:	100 10 %	40 4 %	50 5 %	0 0 %	✓ ΣΔ%: 19 %
Sum Of Development Resources:	Σ%:	11 %	6 %	6 %	3 %	
Value To Cost:		21.90	39.20	40.70	12.00	
Ratio (Worst Case)		18.80	25.80	21.60	7.30	
Ratio (Cred. - adjusted)		5.00	14.50	13.50	0.70	
Ratio (Worst Case Cred. - adjusted)		292.00	113.40	22.50	2.80	

Comments:

Figure 12.3.1.12 A



IE can be used for a wide variety of purposes including:

1. Evaluating a single design idea. How good is the idea for us?
2. Comparing two or more design ideas to find a winner, or set of winners. Hint: Use IE, if you want to set up an argument against a prevailing popular, but weak design idea!
3. Gaining an architectural overview of the impact of all the design ideas on all the objectives and budgets. Are there any negative side effects? What is the cumulative effect?
4. Obtaining systems engineering views of specific components, or specific performance aspects. For example: Are we going to achieve the reliability levels?
5. Analyzing risk: evaluating a design with regard to 'worst case' uncertainty and minimum credibility.
6. Planning evolutionary project delivery steps with regard to performance, value, benefits and cost.
7. Monitoring, for project management accounting purposes, the progress of individual evolutionary project delivery steps and, the progress to date compared against the requirement specification or management objectives.
8. Predicting future costs, project timescales and performance levels.
9. Understanding organizational responsibility in terms of performance and budgets by organizational function.*
10. Achieving rigorous quality control of a design specification prior to management reviews and approval.
11. Presenting ideas to committees, management boards, senior managers, review boards and customers for approval.
12. Identifying which parts of the design are the weakest (risk analysis). Hint: If there are no obvious alternative design ideas, any 'weak links' should be tried out earliest, in case they do not work well (risk management). This impacts scheduling.
13. Enabling configuration management of design, design changes, and change consequences.
14. Permitting delegation of decision-making to teams. People can achieve better internal progress control using IE, than they can from repeatedly making progress reports to others, and acting on others' feedback.
15. Presenting overviews of very large, complex projects and systems by using hierarchical IE tables. Aim for a one page top-level IE view for senior management.
16. Enabling cross-organizational co-operation by presenting overviews of how the design ideas of different projects contribute towards corporate objectives. Any common and conflicting design ideas can be identified. Hint: This is important from a customer viewpoint; different projects might well be delivering to the same customer interface.
17. Controlling the design process. You can see what you need, and see if your idea has it by using an IE table. For example, which design idea contributes best to achieving usability? Which one costs too much?
18. Strengthening design. You can see where your design ideas are failing to impact sufficiently on the objectives; and this can provoke thought to discover new design ideas or modify existing ones.
19. Helping informal reasoning and discussion of ideas by providing a framework model in our minds of how the design is connected to the requirements.
20. Strengthening the specified requirements. Sometimes, you can identify a design idea, which has a great deal of popular support, but doesn't appear to impact your requirements. You should investigate the likely impacts of the design idea with a view to identifying additional stakeholder requirements. This may provide the underlying reason for the popular support. You might also identify additional types of stakeholders.

*Note: * In 1992, Steve Poppe pioneered this use at executive level while at British Telecom, North America.*

Figure 9.2
Purposes for the use of Impact Estimation. IE can have a wide variety of uses for a systems

Oh yes, 'Planning Brexit', too (true!)

Or at least making it clear that it was no simple matter.

12.3.1.16 Estimating Future Levels of Value Quantities. Principles of IET.

1. **THINK**: Impact Estimation (IE) makes us think, research, and present; much more objectively and clearly, about any type of *means* (strategies, options, designs, architecture, solutions).

2. **QUANTIFY-V X QUANTIFY-S**: IE combines two major quantification ideas: the *quantification of all critical stakeholder values*, and the *quantification the impact of solution attributes, on those stakeholder values*.

3. **OVERVIEW**: IE being completely numeric, means that we can compute a number of interesting 'overview numbers', such as the *overall values for costs with regard to risks*, and the *safety margins for the overall set of ideas*.

4. **RISKS**: The IE Table allows us to specify risks, see risks, cumulate overall risks, and to prioritize *based on* risks.

5. **CREDIBILITY**: the detailed collection of *Evidence* for an estimate, and the *source* of the evidence, can be turned into a *Credibility number*. This makes us able to see *how risky* it is to believe the evaluations of the IE model. We can even look at the *average credibility rating* for 100 or more cells on a 10x10 table.

6. **AUDIT**: any type of audit, review, or Quality Control, of any plan, is structurally made much-easier, by organizing the plan into an Impact Estimation Table (IET).

7. **PRIORITY**: The IET gives us a systematic set of data for *initially*, and then *iteratively*, **prioritizing** the agile implementation of sub-solutions, based on their cost-effectiveness, and risks. No *fixed subjective* weights!

8. **LEVELS**: You can model 'any size of *complexity* of system', using a hierarchical set of IE Tables. Usually starting at the top with about 10x10, or a page, then decomposing values, and strategies, as needed, for detail.

9. **ACCOUNTING:** An IT Table can be used as a project **budget**, and then measurement of values and costs delivered incrementally can be used as the **'accounting system'** for progress of the project.

10. **AI:** the structure, defined concepts, and quantification in an IE Table, and in Planguage, is a basis for *artificial intelligence* of many sorts, for example *automated desig*n or architecture; and automated Quality Control.

This was conceptually pioneered, with still-working apps, by Lech Krzanik, Aspect Engine, on an Apple II in Forth, for his PhD, in late 1970's [B15, 1988]. We think we are ready to move forward on this, with the advent of AI, Big Data, Internet, ValPlan and GraphMetrix tools.

12.3.1.16

The Impact Estimation Principle:

Any solution's effectiveness can be quantified for any critical stakeholder-value quantified-requirements.

We do not need opinions.
We do not need soft undefined values.
We do not need assertions without evidence
We do not need suggested solutions without responsible people.
We *can* think like engineers about *large complex systems,* which must use *engineering thinking* to avoid failure.

13. Background Specifications

Chapter 13.0 Background Specifications.

‘Background’ Planguage specifications are added to the core requirement (the ‘really required’ stuff), and intermixed with those ‘core’ requirements (Scale, Tolerable, Goal).

Background specs are part of a SEA Language (Planguage) culture which believes that rich ‘written specs’ beat ‘human memory’. **Digital** written information persists better, and allows ‘smarter’ apps.

This is true, as projects scale up, and become geographically decentralized. Simple ‘local group’ methods (yellow stickies, standup retrospectives) do not work any more.

People have to find project information, and share their own project-information *wherever* they are, and *whenever* they are ready to do so.

Using *written* info, and especially *digital* info is the right direction.

Specifications are not simply drafted once, they *accumulate*, over time, from *many* sources, and *much* feedback and *learning*. We need to deal with the dynamics of this, digitally.

At the same time, even in the largest of projects, there is a right time for video face-to-face meetings, to build trust, motivate, and ferret out project info, that itself needs writing down, to share with everybody, sooner or later, if useful.

One data-detail can be the difference between project or value failure and success.

Capturing *digitally* is cheap, compared to forgetting a critical detail.

I think it is important to distinguish between old-fashioned written cultures (paper, copies, issued infrequently, available to few), and a **digital written culture** (internet, app based, continuously updated, from anywhere on the planet, structured for automated analysis, connected to intelligible data sets).

Some of the prejudices against written bureaucratic cultures (and quite right these negative reactions were) were based on the 1990s pre-internet experiences. 'Written' is *not* now, what it was *then*.

Our Planguage/SEAL requirements ideas grew up, using word processors and spreadsheets, but before the internet.

By looking at the capability and potential of the ValPlan.net app, we can see a powerful current capability, and also a potential, for well-structured, well-defined requirements (and other architecture project specifications).

We detest unnecessary bureaucracy! But *some* degree of 'bureaucracies' payoff, and we have to know the difference.

Background specs help manage Stakeholder Relationships

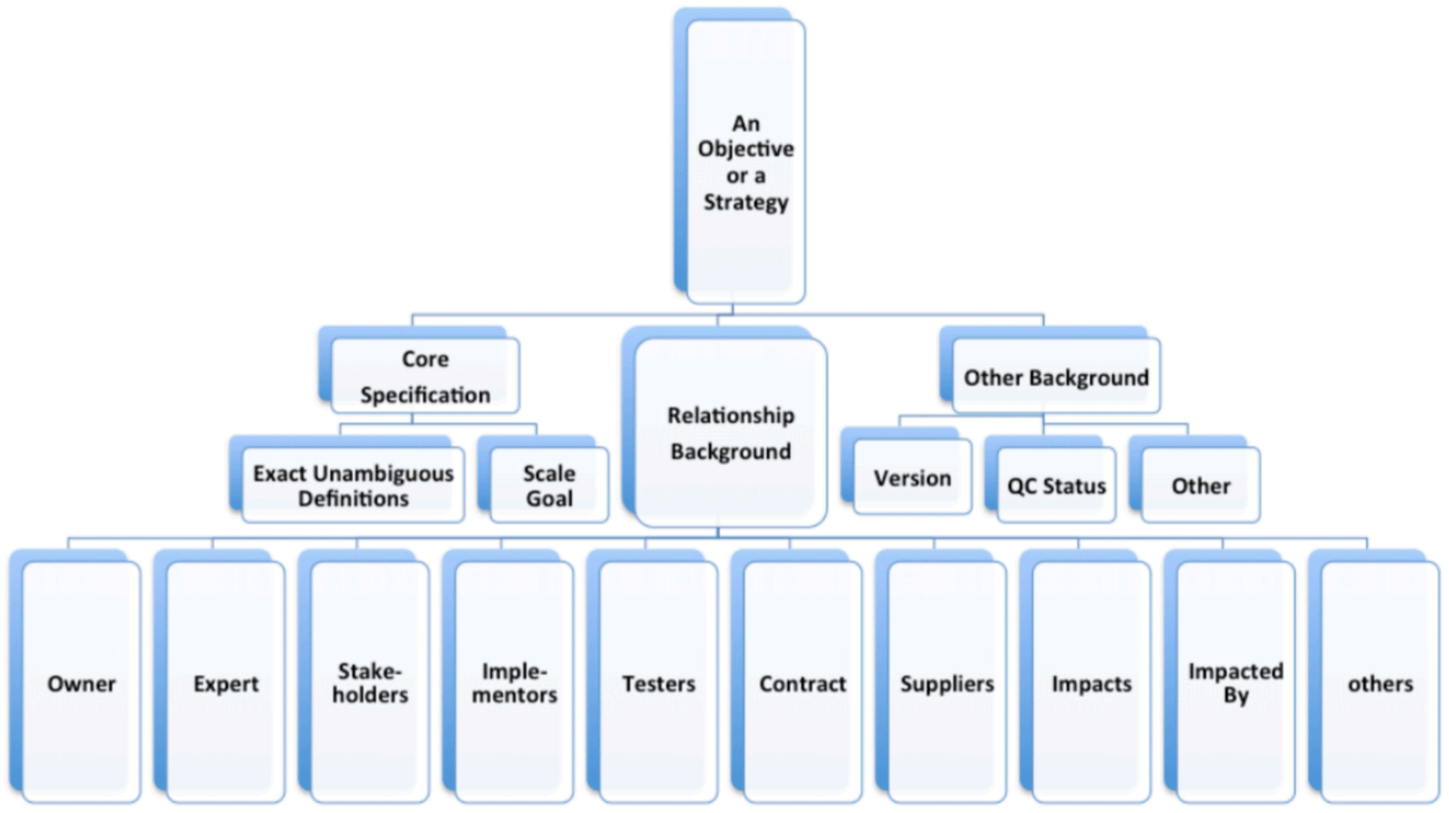
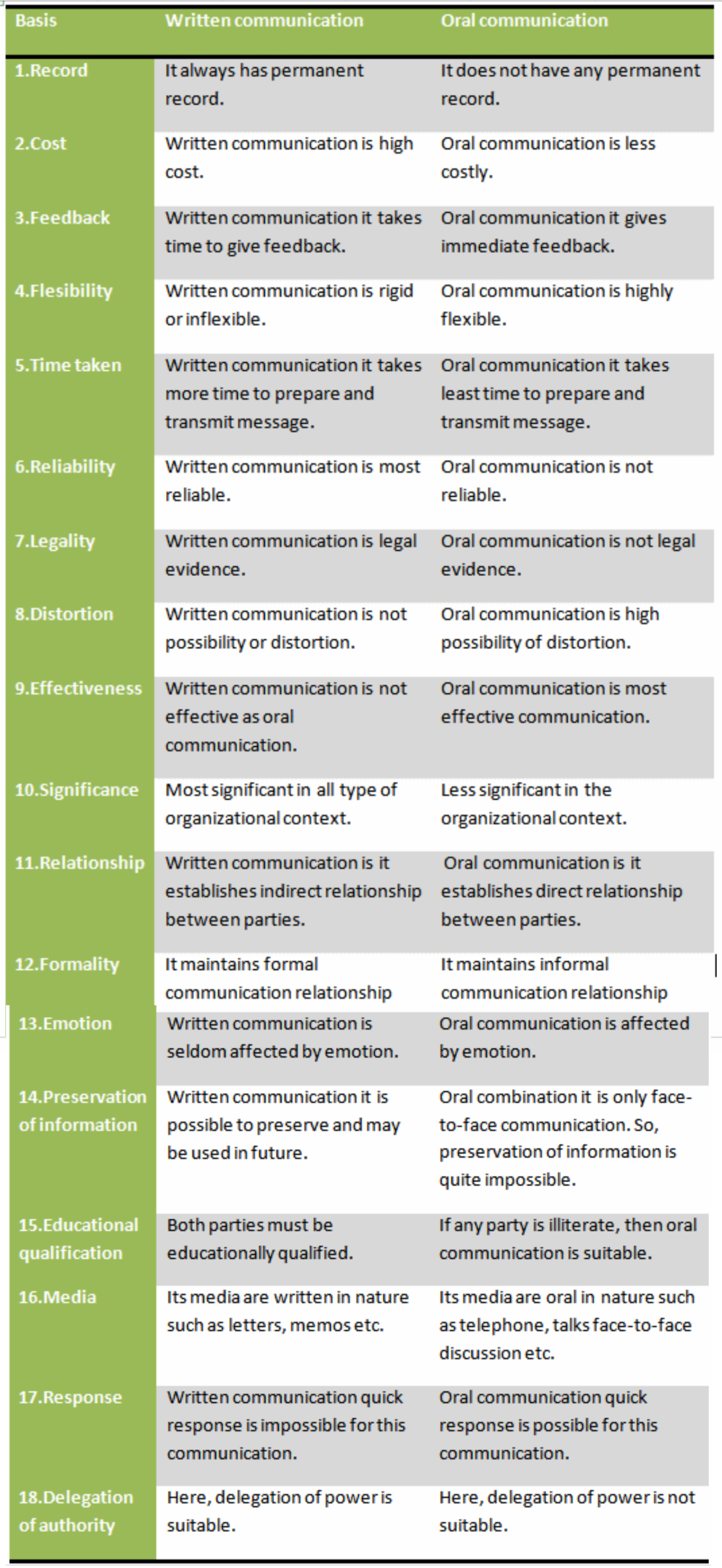


Figure 3.1 B. Some common relationship parameters in a specification.

Figure 13.0 Source [VP] Fig. 3.1



Basis	Written communication	Oral communication
1.Record	It always has permanent record.	It does not have any permanent record.
2.Cost	Written communication is high cost.	Oral communication is less costly.
3.Feedback	Written communication it takes time to give feedback.	Oral communication it gives immediate feedback.
4.Flesibility	Written communication is rigid or inflexible.	Oral communication is highly flexible.
5.Time taken	Written communication it takes more time to prepare and transmit message.	Oral communication it takes least time to prepare and transmit message.
6.Reliability	Written communication is most reliable.	Oral communication is not reliable.
7.Legality	Written communication is legal evidence.	Oral communication is not legal evidence.
8.Distortion	Written communication is not possibility or distortion.	Oral communication is high possibility of distortion.
9.Effectiveness	Written communication is not effective as oral communication.	Oral communication is most effective communication.
10.Significance	Most significant in all type of organizational context.	Less significant in the organizational context.
11.Relationship	Written communication is it establishes indirect relationship between parties.	Oral communication is it establishes direct relationship between parties.
12.Formality	It maintains formal communication relationship	It maintains informal communication relationship
13.Emotion	Written communication is seldom affected by emotion.	Oral communication is affected by emotion.
14.Preservation of information	Written communication it is possible to preserve and may be used in future.	Oral combination it is only face-to-face communication. So, preservation of information is quite impossible.
15.Educational qualification	Both parties must be educationally qualified.	If any party is illiterate, then oral communication is suitable.
16.Media	Its media are written in nature such as letters, memos etc.	Its media are oral in nature such as telephone, talks face-to-face discussion etc.
17.Response	Written communication quick response is impossible for this communication.	Oral communication quick response is possible for this communication.
18.Delegation of authority	Here, delegation of power is suitable.	Here, delegation of power is not suitable.

Figure 13.0.1. A Comparison of Written and Oral Communication Attributes. https://thebusinesscommunication.com/difference-between-oral-and-written-communication/

I excuse the poor grammar and spelling!

It is a full set of ideas.

And it makes the point that the written/oral comparison has many factors to consider.

Even *before* we look at digitization.

Most of the 'Basis' attributes could do with a Scale definition.

And then we could numerically evaluate the modes of communication (after they too were better-defined and technologically updated).

13.1 The *General* Purposes of Background Specifications.

The 'Background Spec' purpose is to *enrich* the requirement spec with information, that

- Might *never* otherwise get specified in *writing*
- Might be '*lost*' in earlier or later documents, like slide presentations and Business Analysis
- Might *not get used seriously* unless they are 'in your face' in the spec.
- Might be *difficult to retrieve* from other documents, or from human memory
- Might be well-known to some, but *unknown* to others
- Might be correct and updated for some people, but *incorrectly remembered and not updated for others*.
- Is needed for ongoing, real time, incremental steps, of value delivery *decision-making*
- Is needed for *risk management, prioritization, taking responsibility, motivation, reviewing efficiently:* and any other purposes on the path to successful value delivery, without being delayed by poor decisions, based on lack of correct information.
- Is needed to enable *automation* of certain aspects of requirements, such as Quality Control, prioritization, presentation, and risk detection. Yes AI. A complete Specification 'Object'. A 'mini' spec object database.
- Background specs help to manage the *updating and changes* to the spec.
- Help us to *follow our adopted defined Rules* (specification standards for requirements).

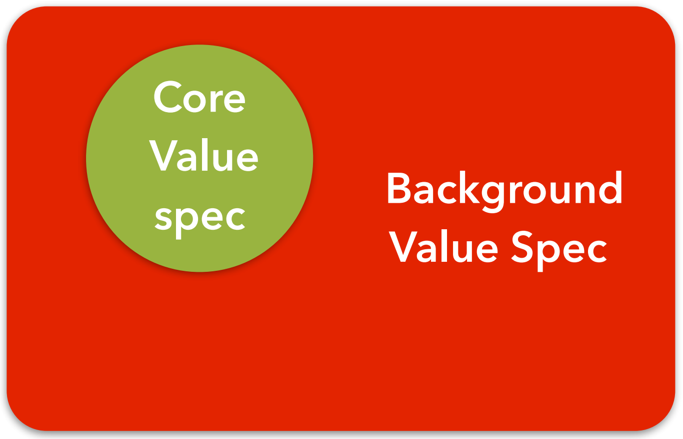
We have already encountered some Background Planguage specs earlier in this book. For example Ambition, Type, Tag, Stakeholders, Status, Level, Past.

User Stories have two kinds of background, built in to their structure: *who* is the stakeholder, and *why* do we need this requirement (Justification, or Rationale).

Good, but not nearly enough

Simple, but 'too simple' for serious purposes in large Enterprise Architecture systems.

Figure13.1.1 : Core Value specification, surrounded by supporting requirement information (background).



The majority of useful architecture planning objects specifications, are <u>not</u> the core specs, they are the <u>Background</u> Specs

13.2 Risk Management with Background Specs

Risk management means, reducing the losses in your scope of work, due to any causes which might somehow be dealt with by better planning, and by better plan specification.

Ericsson of Sweden had a deep insight when in their Quality Policy (see quote in VP Book) they declared that risk management is the job of every engineer, at all times.

I also believe that in the area of requirements, every little detail of specification has a potential of unleashing, or ignoring, risks.

Risks are usually very much larger in consequence, than any cost associated with reducing the risks. We do risk management by having more-solid requirements *craft*-capability. Attention to detail.

Let me be more direct. I believe that every detail in *this* book, are potential tools for reducing risks, systematically.

This is not a co-incidence. I am by nature a very risk-adverse person, so I *designed* these methods to deal with requirements-and-systems risks.

Let me use as an example: a major idea in this book.

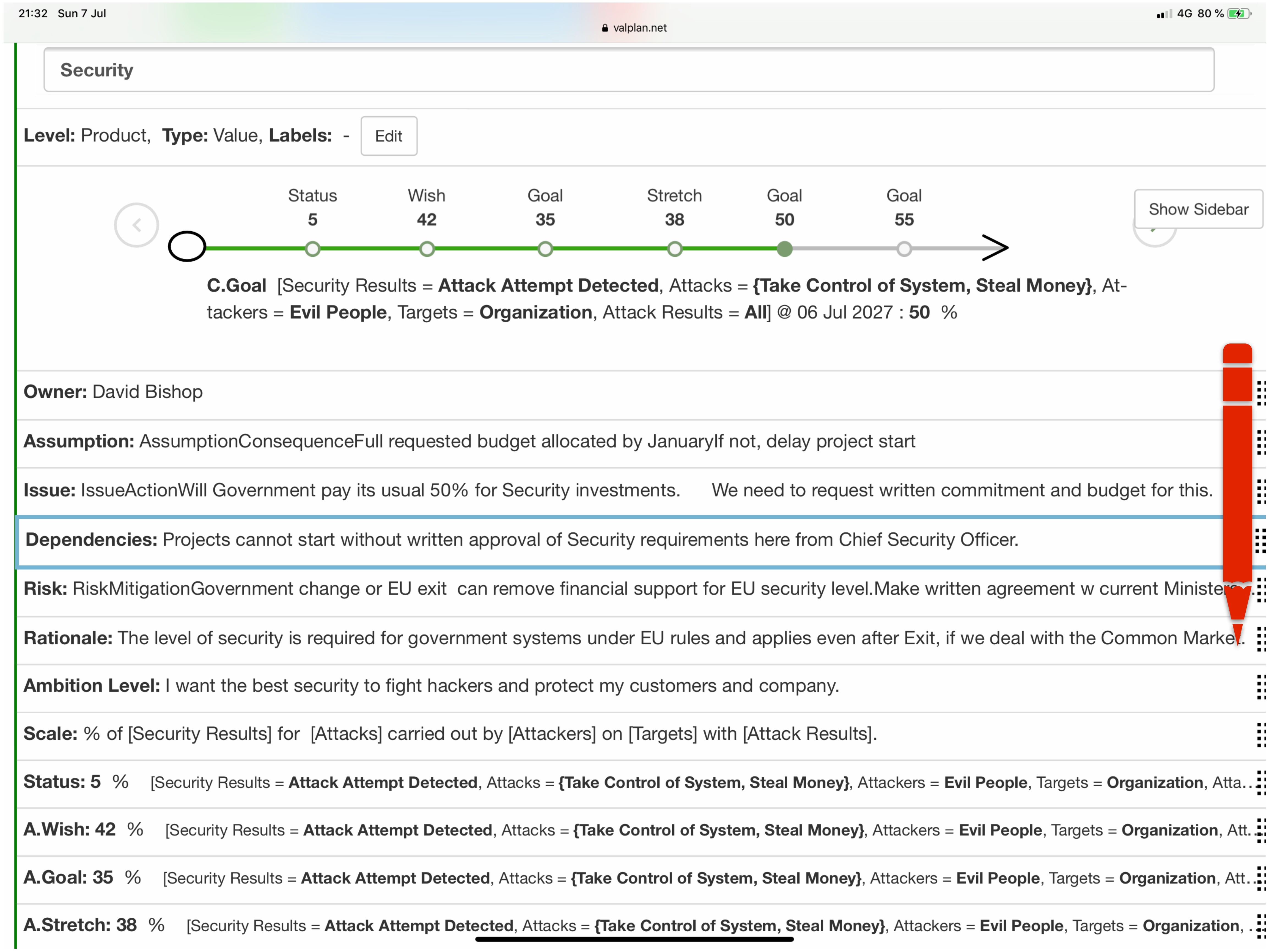
Quantification of the Value requirements.

- If a Value Requirement is ***not* quantified,**
- you *immediately* incur a huge and unnecessary **risk, because**
  - Nobody can understand the requirement, in the same way.
  - People will waste time and money working towards their private interpretation of the requirement ('better security').
  - At worst, the entire project can fail for this one thing alone (plenty of projects fail now) [0.0]

Some other scattered examples of Background spec details, related to Risks

- If the requirement is not **tagged**, giving it clear-long term identification, then it might be missed in test planning, and be delivered in failed condition.
  - Too many *value requirements* are just *bullet points* on a slide
  - Tagging a spec, means we can reuse it, and avoid the risk of different non-identical-copies, or versions, of the same spec.
- If we do not give the **Source** of a requirement, which stakeholders want it (and why), then quality control of its current validity, is less likely, and bad requirements might get implemented.
- If we do not capture the **Ambition Level** (the blah blah) and its <u>source</u>, then we risk losing alignment with higher-priority objectives, and their possible changes, like from new management.
- If we do not assign a **Specification Owner** to each single requirement, there is a high risk that no one will be qualified and motivated, to keep the spec properly updated, and to keep it high quality.
- An ‘orphan’ specification, without an Owner.

Hopefully you can guess how such specs help us to see and manage risks



21:32 Sun 7 Jul

valplan.net

4G 80 %

Security

Level: Product, **Type:** Value, **Labels:** - Edit

C.Goal [Security Results = **Attack Attempt Detected**, Attacks = **{Take Control of System, Steal Money}**, Attackers = **Evil People**, Targets = **Organization**, Attack Results = **All**] @ 06 Jul 2027 : **50** %

Owner: David Bishop

Assumption: AssumptionConsequenceFull requested budget allocated by JanuaryIf not, delay project start

Issue: IssueActionWill Government pay its usual 50% for Security investments. We need to request written commitment and budget for this.

Dependencies: Projects cannot start without written approval of Security requirements here from Chief Security Officer.

Risk: RiskMitigationGovernment change or EU exit can remove financial support for EU security level.Make written agreement w current Ministe...

Rationale: The level of security is required for government systems under EU rules and applies even after Exit, if we deal with the Common Market.

Ambition Level: I want the best security to fight hackers and protect my customers and company.

Scale: % of [Security Results] for [Attacks] carried out by [Attackers] on [Targets] with [Attack Results].

Status: 5 % [Security Results = **Attack Attempt Detected**, Attacks = **{Take Control of System, Steal Money}**, Attackers = **Evil People**, Targets = **Organization**, Atta...

A.Wish: 42 % [Security Results = **Attack Attempt Detected**, Attacks = **{Take Control of System, Steal Money}**, Attackers = **Evil People**, Targets = **Organization**, Att...

A.Goal: 35 % [Security Results = **Attack Attempt Detected**, Attacks = **{Take Control of System, Steal Money}**, Attackers = **Evil People**, Targets = **Organization**, Att...

A.Stretch: 38 % [Security Results = **Attack Attempt Detected**, Attacks = **{Take Control of System, Steal Money}**, Attackers = **Evil People**, Targets = **Organization**, ...

Figure 13.2. some Background Specifications have been added to the Security specification. Owner, Assumption, Issue, Dependencies, Risk, Rationale

13.3 Prioritization using background specs

In my 'Evolutionary Value Delivery' culture, all Value requirements are not equal. We are going to start a stream of value-improvement deliveries. So we have to figure out which Value deliveries are smartest to deliver, *early*.

There is no simple method to help us decide what to do first or next, which will be realistic, and to give the most satisfactory results. There are far too many different dynamically-changing factors, which influence a prioritization decision.

So, our best suggested approach today is to collect *prioritization*-information, directly in the *requirement* specification. This background information can be used to help you decide *which* Value levels to prioritize.

A simple example might be that Value X has 3 Stakeholders interested, one of whom is the Government, and Value Y has has 2 stakeholders interested, one of whom is your boss.

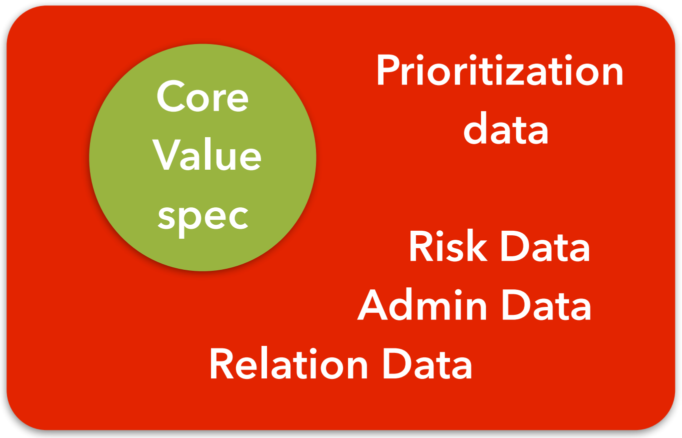
You cannot deliver *both*, in the coming time period, you must choose one. What would your boss advise?

Wait, it is never so simple! Value X delivery has an estimated return of 300% annually, and Value Y has only 120%.

The bad news is that there are many *more* factors to consider in this case.

You can always simplify, and ignore those factors. But sooner or later, somebody is going to ask why Factor XYZ was not considered ('Sales with Potentially large new customers', for example).

Figure13.3 : Core Value specification, surrounded by supporting requirement information (background).



The majority of useful architecture planning objects specifications, are <u>not</u> the core specs, they are the <u>Background</u> Specs

An automated sort, best option at left, for delivering the greatest *set* of values, at the lowest *set* of costs, with regard to risks and uncertainty.

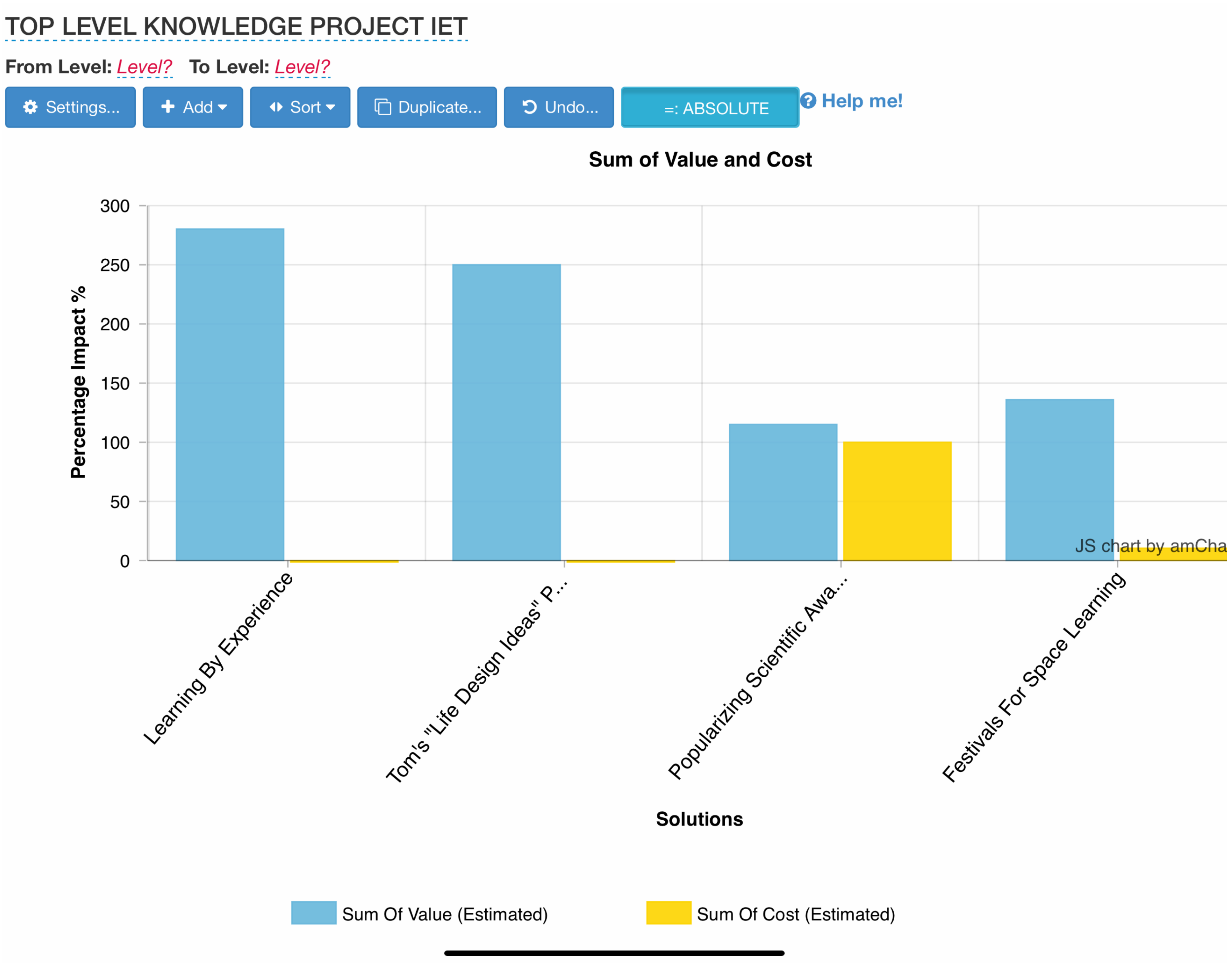
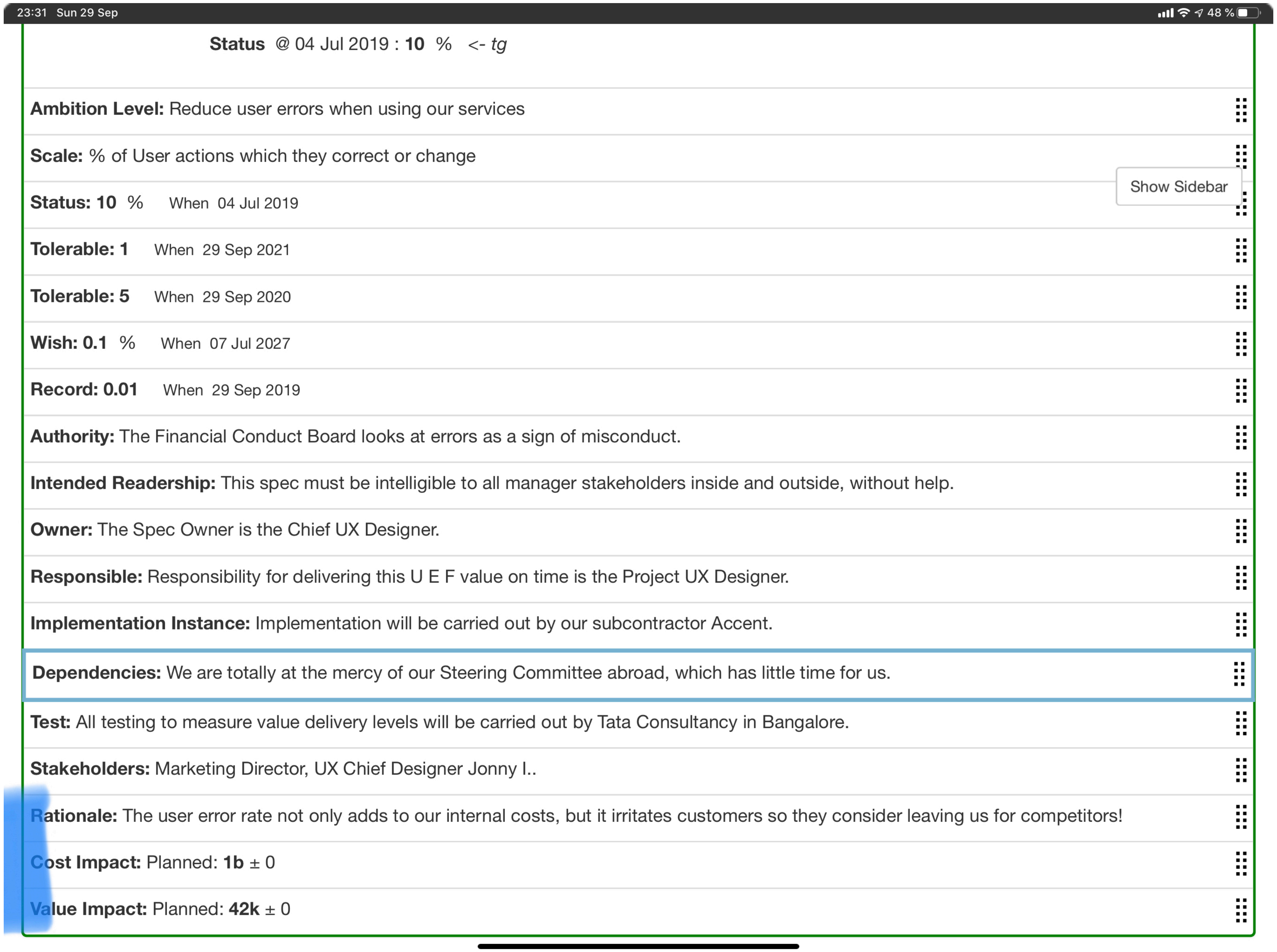


Figure 13.3 : Source: Masterclass, Katowice, exercise on spreading knowledge nationally. 2018. This chart uses data in the Value Specification, along with a lot of other factors to optimize the flow of value for money.

Adding prioritization parameters, to the 'objective' specification object.



23:31 Sun 29 Sep 48 %

Status @ 04 Jul 2019 : **10** % <- *tg*

Ambition Level: Reduce user errors when using our services

Scale: % of User actions which they correct or change

Show Sidebar

Status: 10 % When 04 Jul 2019

Tolerable: 1 When 29 Sep 2021

Tolerable: 5 When 29 Sep 2020

Wish: 0.1 % When 07 Jul 2027

Record: 0.01 When 29 Sep 2019

Authority: The Financial Conduct Board looks at errors as a sign of misconduct.

Intended Readership: This spec must be intelligible to all manager stakeholders inside and outside, without help.

Owner: The Spec Owner is the Chief UX Designer.

Responsible: Responsibility for delivering this U E F value on time is the Project UX Designer.

Implementation Instance: Implementation will be carried out by our subcontractor Accent.

Dependencies: We are totally at the mercy of our Steering Committee abroad, which has little time for us.

Test: All testing to measure value delivery levels will be carried out by Tata Consultancy in Bangalore.

Stakeholders: Marketing Director, UX Chief Designer Jonny I..

Rationale: The user error rate not only adds to our internal costs, but it irritates customers so they consider leaving us for competitors!

Cost Impact: Planned: **1b** ± 0

Value Impact: Planned: **42k** ± 0

Figure 13.4 : we added 4 examples of background Parameters to the User Error Frequency Value spec. Hopefully you can see that each one might contribute to a prioritization decision.

See: Stakeholders, Cost Impact, Value Impact, and Rationale

13.4 Responsibility and Motivation with Background specs

We now can see some more information about the roles and power of the stakeholders involved for this one value.

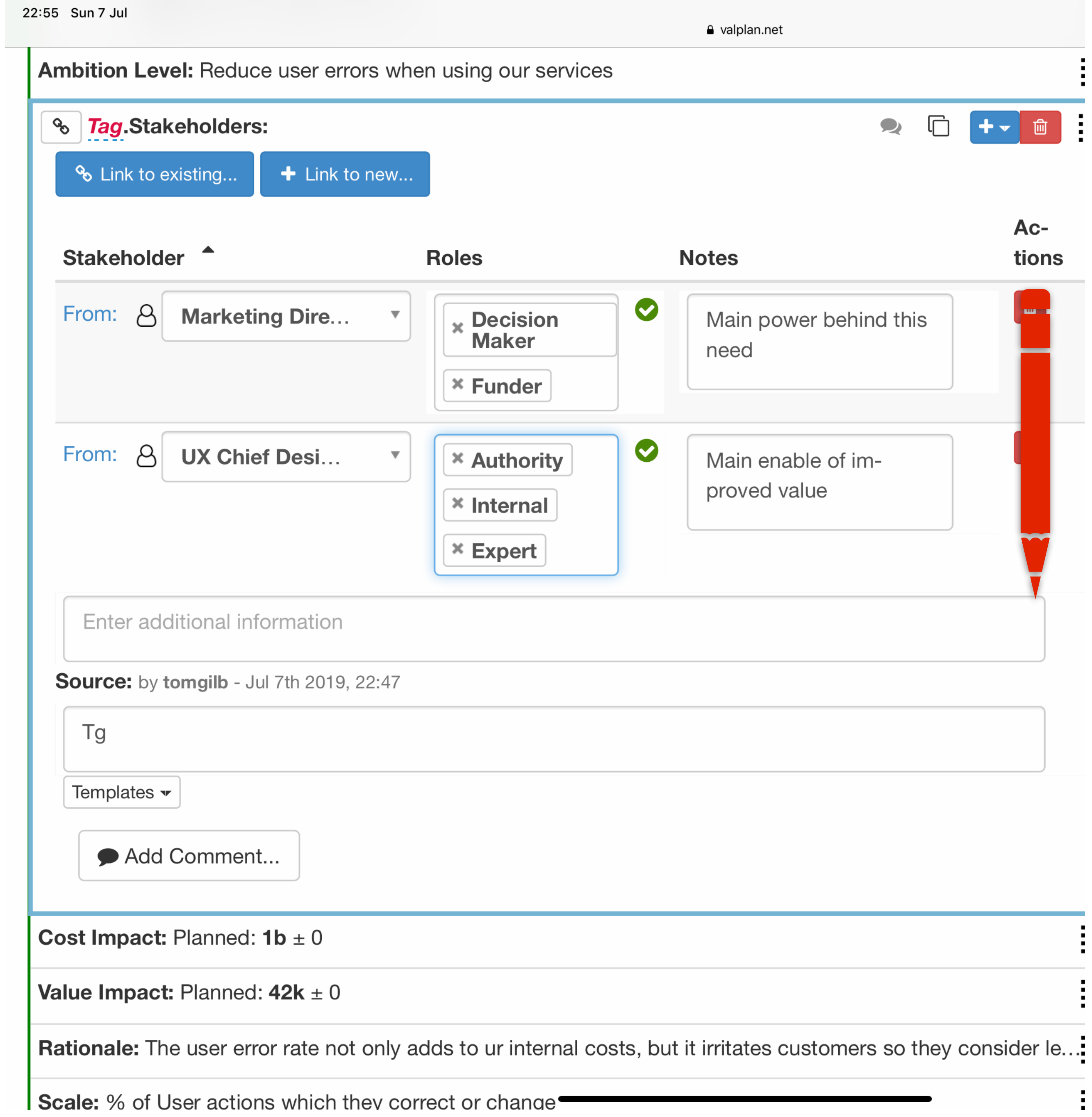
That information is even more useful than just knowing the *names* of the stakeholders.

There is already enough digital information in the 'Roles' categories to help us prioritize this value better automatically.

Would you prioritize *Funders* over *Authorities*?

Here, below, is a set of Planguage parameters added to the *User Error Frequency* value requirements spec. [Figure 13.5] These Background Value Spec Parameters clarify and assign formal responsibility for different aspects of the value requirement.

It is worth pointing out that we now have a practical tool for decentralizing authority, for delegating authority, to people and groups who are interested in having it, who have or *will make* the time to do things properly, and are specialist in this area.



22:55 Sun 7 Jul

valplan.net

Ambition Level: Reduce user errors when using our services

Tag.**Stakeholders:**

Link to existing... | Link to new...

Stakeholder		Roles	Notes	Ac-tions
From:	Marketing Dire...	Decision Maker; Funder	Main power behind this need	
From:	UX Chief Desi...	Authority; Internal; Expert	Main enable of im-proved value	

Enter additional information

Source: by **tomgilb** - Jul 7th 2019, 22:47

Tg

Templates

Add Comment...

Cost Impact: Planned: **1b** ± 0

Value Impact: Planned: **42k** ± 0

Rationale: The user error rate not only adds to ur internal costs, but it irritates customers so they consider le...

Scale: % of User actions which thev correct or change

Figure 13.5 : This is a *detailed window* for the Stakeholders spec, for the 'User Error Frequency' value spec, summarized above.

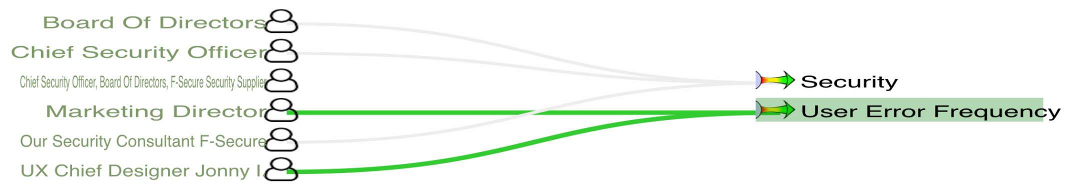
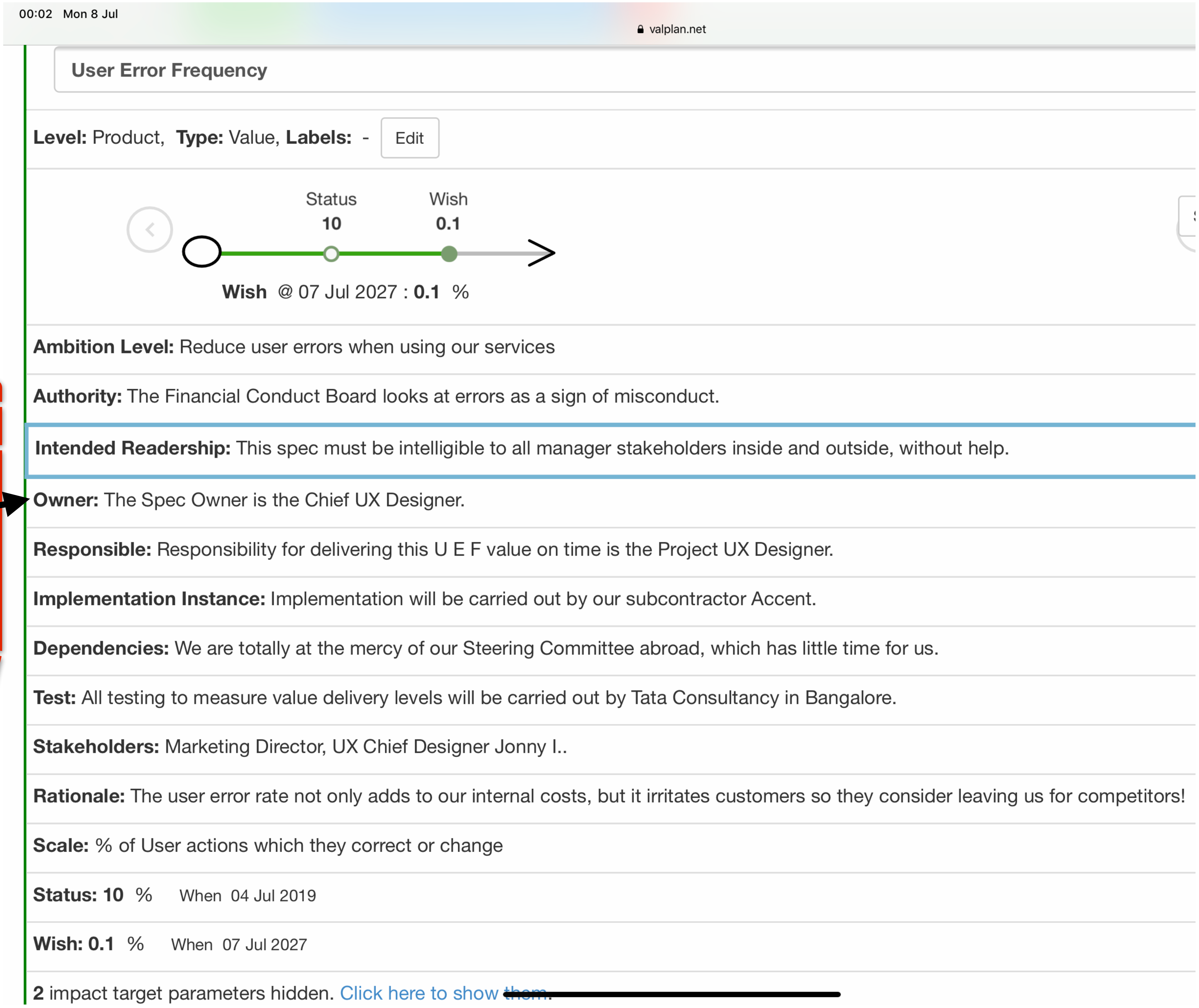

Board Of Directors
Chief Security Officer
Chief Security Officer, Board Of Directors, F-Secure Security Supplier
Marketing Director
Our Security Consultant F-Secure
UX Chief Designer Jonny I.
Security
User Error Frequency

Figure 13.6 : the relationship between stakeholders and values is digital, so we can display it graphically (automatic thick lines to show relationships). Useful when things get voluminous and complicated.

I have observed that the moment you put someone's name on a responsibility voluntarily, it gets taken quite seriously. It is their baby and their honor! (See the **Owner** spec. Here Figure 13.7)

This Chapter 13 , on Background Specifications, was largely borrowed from my book Value Requirements ([VR] 2019)



User Error Frequency

Level: Product, **Type:** Value, **Labels:** -

Wish @ 07 Jul 2027 : **0.1** %

Ambition Level: Reduce user errors when using our services

Authority: The Financial Conduct Board looks at errors as a sign of misconduct.

Intended Readership: This spec must be intelligible to all manager stakeholders inside and outside, without help.

Owner: The Spec Owner is the Chief UX Designer.

Responsible: Responsibility for delivering this U E F value on time is the Project UX Designer.

Implementation Instance: Implementation will be carried out by our subcontractor Accent.

Dependencies: We are totally at the mercy of our Steering Committee abroad, which has little time for us.

Test: All testing to measure value delivery levels will be carried out by Tata Consultancy in Bangalore.

Stakeholders: Marketing Director, UX Chief Designer Jonny I..

Rationale: The user error rate not only adds to our internal costs, but it irritates customers so they consider leaving us for competitors!

Scale: % of User actions which they correct or change

Status: 10 % When 04 Jul 2019

Wish: 0.1 % When 07 Jul 2027

2 impact target parameters hidden. Click here to show them.

Figure 13.7 : adding specific responsibilities, as Background statements, to a Value requirement.

References

R. Architecture References Others

R. Marcin Ros, **Enterprise Architecture as a System Engineering Discipline,** https://medium.com/@mikolunar/enterprise-architecture-as-a-system-engineering-discipline-4d1ffff3fb08

R2. Kurek, Johnsen, Mulder, **Measuring the value of Enterprise Architecture on IT projects with CHAOS Research, Jul 10, 2017,** https://pdfs.semanticscholar.org/96a5/2c679cd1dc2c9691ec8e92f9d3066089888e.pdf

R3.Michael Keeling, **Dealing with Constraints in Software Architecture Design**, - October 22, 2014, https://www.neverletdown.net/2014/10/dealing-with-constraints-in-software-architecture.html

R4. Bass, Clements, Kazman, **Software Architecture in Practice**, 2014 Sample Chapter on Quality Requirements., http://ptgmedia.pearsoncmg.com/images/9780321815736/samplepages/0321815734.pdf

Note on R4. I would differ with the ideas expressed here. But the reader can compare for themselves, this book and this trio's ideas on function, qualities, constraints and matching to architecture. I'd be happy to detail my dissension if someone thinks it has a use. TG Hint: they do not understand quantified qualities at all, + more.

Gilb Books Overview Page with Mnemonic code

~~VE: **Vision Engineering**.~~

("Value Planning: Top Level Vision Engineering")
How to communicate critical visions and values quantitatively. Using The Planning Language.
 http://concepts.gilb.com/dl926
A 64 Page pdf book. Aimed at demonstrating with examples how top management can communicate their 'visions' far more clearly.

VP: **"Value Planning. Practical Tools for Clearer Management Communication"**
Digital Only Book. 2016-2019, 893 pages, €10
https://www.gilb.com/store/2W2zCX6z
T*his book is aimed at management planning. It is based on the Planguage standards in 'Competitive Engineering' (2005). It contains detailed practical case studies and examples, as well as over 100 basic planning principles.*

Summer 2019 Free Digital Books https://www.dropbox.com/sh/adcrki52xo5zb36/AABMD_2GOX4rT6c-HRCmT-Qua?dl=0
SP: Sustainability Planning: https://tinyurl.com/UNGoalsGilb
VR: Value Requirements: https://tinyurl.com/ValueRequirementsBook
VM: Value Management: https://tinyurl.com/ValueManagementBook
VA: Value Agile: https://tinyurl.com/ValueAgileBook
VD: Value Design: https://tinyurl.com/ValueDesignBook
Slides: http://concepts.gilb.com/dl972
VIDEOS (5 BOOKS): https://www.gilb.com/blog/Agile-Tools-for-Value-Delivery-by-Tom-Gilb (these are short courses, 2-3 hours each)

Summer 2020 Free Digital Books, https://www.dropbox.com/sh/tj1p6a3omlg9hx3/AABXuj_YnUmAFeRWOpGVvQtIa?dl=0
Q: **QUANTeer: The Art of quantifying your value ideas**., https://tinyurl.com/Quanteer
P: **Planalysis: Analyzing Garbage Specs**. https://tinyurl.com/PLanalysisFree
G: **Governeering:Government Systems Engineering Planning.**
https://tinyurl.com/Governeering
KEN: Knowledge Edu-Neering, https://tinyurl.com/KENGilb

12?: This Book
'12?: Twelve Tough Questions for Better Management'
https://tinyurl.com/12TOUGH

Summer 2018 Books, for Sale at **gilb.com**
https://www.gilb.com/store?tag=books
T: Technoscopes
IC: Innovative Creativity
100: 100 Planning Principles
L: Life Design
C: Clear Communication

CE: Gilb, Tom, "Competitive Engineering', 2005, Elsevier
https://www.gilb.com/p/competitive-engineering (free pdf)
Or paper edition: https://www.amazon.com/dp/0750665076/ref=rdr_ext_sb_ti_sims_2
The '12 Tough Questions' were printed in this book. Page 7-8. And based on a years earlier paper.

PoSEM: Gilb: Principles of Software Engineering Management, Addison Wesley Longman, 1988
Chapt 15 Deeper Perspectives on Evolutionary Delivery
www.gilb.com/dl561 **(free, agile history)**
https://www.amazon.com/Principles-Software-Engineering-Management-Gilb/dp/0201192462

R.ValPlan: VALPLAN INFO
https://www.gilb.com/valplan, actual app is at ValPlan.net
I should declare a personal interest in this company.
10.1. Based on my ideas [B1]. 2. Our company is marketing it.
Richard Smith, UK, is our developer/designer hero building it.

B Gilb Books. Page 1A

Main Current Books Written by Tom Gilb. Supports this book with detail.

B1.CE: **Competitive Engineering** (paper or digital 2005).

The **definition of the Planguage. A Handbook and a Planguage standard.**

https://www.gilb.com/p/competitive-engineering (free pdf)

and paper via Amazon (Kindle and paper)

https://www.amazon.com/dp/0750665076/ref=rdr_ext_sb_ti_sims_2

B2:VP: **Value Planning**

“Value Planning. **Practical Tools for Clearer Management Communication”**

Digital Only Book. 2016-2019, 893 pages, €10

https://www.gilb.com/store/2W2zCX6z

This book is aimed at management planning. It is based on the Planguage standards in ‘Competitive Engineering’ (2005). It contains detailed practical case studies and examples, as well as over 100 basic planning principles.

The ‘Technoscopes’ book (2018) is a condenses version of this with the 100 principles and some examples o quotes related to the principles.

The ‘Vision Engineering’ book [B3, VE] is. Short (60 pages) top manager oriented overview of the ideas in Value Planning, and it is the **front end (the real book)** of the Value Planning book.

B2.**Decomposition**: https://tinyurl.com/VPDecomposition

B2: **Prioritization**: https://www.dropbox.com/sh/34llx1a7ckyagxl/AAA0pDzSxN5WmoP9lOKR0Mpca?dl=0

P18: **Risk Management Chapter 7, (70 pages) VP Value Planning** Book. https://tinyurl.com/RiskMgtVP

B2: Chapter 3 **Levels of interest,** https://www.dropbox.com/sh/xbzn5s8imf9vlao/AAB8h-OFvQmJ_w3wNhrDxa9_a?dl=0

B2: Chapter 8 **Delegation Outsourcing Contracting,** https://www.dropbox.com/sh/eubo1zkvybl2q8k/AAD6cUUIOqcooaTPK2OZx6gua?dl=0

B3:VE.**Vision Engineering**.

“Value Planning: **Top Level Vision Engineering”**

How to communicate critical visions and values quantitatively. Using The Planning Language.

http://concepts.gilb.com/dl926

A 64 Page pdf book. Aimed at demonstrating with examples how top management can communicate their ‘visions’ far more clearly.

B Gilb Books. Page 1B

Main Current Books Written by Tom Gilb. Supports this book with detail.

B4. LINK TO **5 NEW DIGITAL BOOKS (B5 to B9)** WRITTEN SUMMER 2019, see also Videos and Slides, same title

See leanpub.com

https://leanpub.com/u/tomgilb

B5:VR. **Value Requirements** book

B6:VD. **Value Design,** Book. July 2019

B7:VM. **Value Management,** book August 2019

B8:VA. **Value Agile ,** 2019

B9SP. **Sustainability Planning,** https://tinyurl.com/UNGoalsGilb, 2019

BOOK, See slides **[P2.2]** http://concepts.gilb.com/dl977

B10. **Books written Summer 2020.**

https://leanpub.com/u/tomgilb

B11 **KEN: Knowledge Edu-Neering booklet**

Leanpub.com/KEN 2020, **CC.**

B12. **Governeering: Government Systems Engineering Planning**: Leanpub.com/Governeering 2020

B13. **PLanalysis: A booklet with advice on how to analyze plans, and make them better,** 2020, Leanpub.com/Planalysis

B14. **QUANTeer: The Art of quantifying your value ideas** Leanpub.com/Quanteer

B15: 12?: **12 Tough Questions for Better Management** leanpub.com/12ToughQuestions

B16. SEA: Systems Enterprise Architecture (SEA) BOOK 2020,.

Leanpub.com/SysEntArchBook

G

Books by Gilb, Page 2

The 2018 5 Books, & Older Gilb Books

G10. **Technoscopes**, 2018

Technoscopes:

Tools for understanding complex projects

https://www.gilb.com/store/Pd4tqL8s

Price €14B12. **Clear Communication**, 2018

G13. **Innovative Creativity**, 2018

‘INNOVATIVE CREATIVITY’ 124 pages €14

https://www.gilb.com/store/QMMQhn2g

G14. **100 Practical Planning Principles**, 2018

Based on the same 100 **Value Planning** sub-sections and principles.

100 Practical Planning Principles. Booklet €14

https://www.gilb.com/store/4vRbzX6X

G15. **PoSEM** 1988, **Principles of Software Engineering Management**, 1988, Pearson.

Chapter 15 Deeper Perspectives on Evolutionary Delivery, www.gilb.com/dl561,

Whole Book (Paper) https://www.amazon.com/Principles-Software-Engineering-Management-Gilb/dp/0201192462

G16. **Software Inspection**, 1993, https://www.amazon.com/Software-Inspection-Tom-Gilb/dp/0201631814

G17. **CLEAR COMMUNICATION Booklet**

“Principles of Clear Communication”

By Tom Gilb

DIGITAL BOOKLET €14

Published 31 August 2018

https://www.gilb.com/store/oJCCxtsM

G18: **Software Metrics**, 1976-7,

Gilb Tom. Software metrics. *Studentlitteratur AB Sweden*, 1976., Tom Gilb. *Software metrics*. Winthrop, 1977. (USA Hardcover). The term **Software Metrics** was coined by me here.

G19. **Life Design**, 2018

LIFE DESIGN Booklet €14

https://www.gilb.com/store/kCBGcG6L

P Papers by Gilb and others

Free Downloadable Papers

P1. 'Agile Project Startup Week', gilb.com/dl568

P2. **Confirmit Case** .http://www.gilb.com/DL32 , 'FROM WATERFALL TO… BY TROND AND TOM GILB

P3.1 Walston, C.E. and Felix, C.P. (1977) **A Method of Programming Measurement and Estimation.** IBM Systems Journal, 16, 54-73. http://dx.doi.org/10.1147/sj.1610.1.0054, $33 Paywall or IEEE., I have a paper copy of this. Tom Gilb, and some of their original data collection schemes they gave me.

P3.2 **'Cleanroom Method',** developed by IBM's Harlan Mills (IBM SJ No. 4/1980)http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan. See here 7.2.3

P3.3 Robert E. Quinnan, **'Software Engineering Management Practices' (Part V),** IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 466~77, https://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan (Quinnan is at end Part 5)

See also [S7] **Technoscopes: Meet the Challenge of Engineering Complexity**, http://concepts.gilb.com/dl968 for Quinnan slides as used in Fig 12.3.2.4

P4:CG: **Full Planguage Concept Glossary,** http://www.gilb.com/dl830

See also [B1] Glossary, and GILB.COM SITE GLOSSARY, http://concepts.gilb.com/A?structure=Glossary&page_ref_id=126

the digital glossary by Kai and company, and ValPlan.net, or other variations of glossary info.

P5. **Agile Specification QC**, in Testing Experience 2009, by Tom Gilb, http://www.gilb.com/DL264

P6. **Estimation: A Paradigm Shift Toward Dynamic Design-to Cost and Radical Management**

Volume 13 Issue 2 of SQP journal - the March 2011 version. http://www.gilb.com/DL460

P7. **Planguage Rules Collection from CE Boo**k.docx,, http://www.gilb.com/dl829, 23 pages., See similar set S3

P8. by Tom Gilb & Kai Gilb, 2018
"All critical outcome value objectives can be quantified and must be."https://medium.com/@kaigilb/principle-quantify-objectives-319a0b9a1f59

P9. **Quantifying Security: How to specify security requirements in a quantified way**. by Tom Gilb, http://www.gilb.com/dl40

P10. **Basic Principles of Security Engineering**., http://concepts.gilb.com/dl948, 2019, 2 Pages 10 principles.

P110.1. **How problems with Quality Function Deployment's** (QFD's) House of Quality (HoQ) can be addressed by **applying some concepts of Impact Estimation (IE)**, http://www.gilb.com/DL119

P12. **Plicons: A Graphic Planning Language for Systems Engineering,** (Plicons Paper), http://www.gilb.com/DL37

P13. "**A Critical Review of Definition of Goals**" , in (Norway, in English), Prosjektledelse 1/2020, http://concepts.gilb.com/dl973, and Widmans Paper I criticize, https://view.joomag.com/prosjektledelse-prosjektledelse-nr-4-2019/0266287001573038589?short

P14. Gilb "**Beyond Scaling: Scale-free Principles for Agile Value Delivery - Agile Engineering",** http://www.gilb.com/dl865 (Paper), (Jan 8 2016). This paper contains considerable detailed systemic explanation as to why the Planguage methods are 'Scale Free'.

P14. **Managing Priorities, A Key to Systematic Decision-making**. With Mark Maier, 2005 (paper), http://www.gilb.com/DL60

P Papers by Gilb and others page 2

Free Downloadable Papers

P15: ISO 31000:2018 (en), Risk management — Guidelines, https://www.iso.org/obp/ui/#iso:std:iso:31000:en

P16: ⍰⍰⍰⍰⍰⍰ ⍰⍰⍰⍰⍰ ⍰⍰⍰⍰⍰⍰⍰José Barateiro et al, **Manage Risks through the Enterprise Architecture**, https://www.researchgate.net/publication/254051828_Manage_Risks_through_the_Enterprise_Architecture. Uses ISO 31000

P17: **Risk Management: A practical toolkit for identifying, analyzing and coping with project risks.** http://www.gilb.com/dl=20

P18: **Risk Management Chapter 7, (70 pages) VP Value Planning** Book. https://tinyurl.com/RiskMgtVP

P19:ST: **Some Stakeholder Slides** 2009-2020, A rough cumulative set as sources. From Gilb and other stakeholder sources, http://www.gilb.com/dl318, Updated last * Sept. 2020

S Slides by Gilb, Free Downloadable

Free Downloadable Slides

S1: PPPP: **Proper Public Planning Principles:** 'Engineering Society', Responsibly

SLIDES = http://concepts.gilb.com/dl980 (pdf) https://tinyurl.com/PPPPslides

Video = https://youtu.be/mIaVLHvQOp0

S2: 'An Agile Project Startup Week'. http://www.gilb.com/dl812

S3. **QC for Design Design Rules from Competitive Engineering** MASTER.key.pdf GilbFest Slides 2015,

http://concepts.gilb.com/dl84, See similar set P7

S4. Most of videos (see below) have a link to their slide set on slide 10.1.

S5.**"Estimation: A Paradigm Shift Toward Dynamic Design-to Cost and Radical Management"**

Slides made for BCS SPA June 1 20110.1. http://www.gilb.com/DL470

S6. **IBM FSD MIlls and Quinnan Slides**. http://concepts.gilb.com/dl896 (see also P3.1 to .3)

S7. **Technoscopes: Meet the Challenge of Engineering Complexity**

SLIDES= http://concepts.gilb.com/dl968. (Several IBM Cleanroom and Quinnan slides here)

VIDEO = https://www.youtube.com/watch?v=920rCFYW3ZQ&list=PLKBhokJ0qd3_wlvr0j85YhmNfNj8ZJ8M-&index=2&t=0s

S8.1 **Using 'Evo' to Rapidly deliver measurable improvements to Aircraft Design Engineering Drawing QC"**

Douglas Aircraft 16 Slides (illustrations missing) Based on cut from paper DL254

http://www.gilb.com/DL253

S8.2 DAC Case Paper

"Using 'Evo' to Rapidly deliver measurable improvements to Aircraft Design Engineering Drawing QC"

McDonnell Douglas Aircraft

Gilb Experience Paper for INCOSE 2002

http://www.gilb.com/DL254

Boeing data is also here and in slides.

(the slides in DL253 are derived from this paper)

S8.3 **"Real Case Aircraft Company Top Level Decision Making for CAD CAM Support Systems"**

FOR McDonnell Douglas Aircraft

Gilb Experience SLIDES (14)

NICE SET WITH ILLUSTRATIONS

http://www.gilb.com/DL255

A good example of analysis of management BS

into Planguage. Reference from Harris for Productivity of Gilb methods.

S8.4 **Boeing Slides, '787'**

March 2008 from Tom and Kai presentation, not sure if on gilb.com, can be supplied by author.. Boeing, Renton studied application of my Inspection methods deeply, and adopted them.

S9. **SERIOUS VALUES CANNOT BE B**S**** , Quantifying AI Transparency,**

and UN Sustainability:

http://concepts.gilb.com/dl962, Aim2North Conf. 7Nov2019, Slides

Podcast video 24 minutes before the lecture, https://www.youtube.com/watch?v=J70zf1gF2b8

S10. **What is Wrong with Balanced Scorecard,** slides, http://concepts.gilb.com/dl135, See https://bscdesigner.com/

S110.1. **10 Suggested Principles for Human Factors Systems Engineering,** http://concepts.gilb.com/dl911, [V14]
Keynote at WUD (Worldwide Usability Day), Silesia, Katowice Poland, 9 Dec. 2017

S12. **ICL CASE Study** from (International Computers Limited), BCS June 12 Lecture 2015, Slides, http://www.gilb.com/dl846

S13. Gilb**. "SCALE-FREE: Practical Scaling Methods for Industrial Systems Engineering"**, lecture slides, http://concepts.gilb.com/dl892, 2016, Considerable citation of Intel experience with Planguage method, by Erik Simmons. Scalability Metrics: and An Engineering Structure, and Principles, for an Agile World for June 5 2018 DND/SINTEF Conference, http:// concepts.gilb.com/dl930. See Scaling paper P14.

S14.1 8. Green Week, The Green Week: Reducing Technical Debt by Engineering, http://www.gilb.com/dl575, May 2013, In agilerecord.com
S14.2 The Green Week Slides http://www.gilb.com/dl660, Smidig/Agile 2013 Oslo

V Video Talks or Courses by Gilb

Videos with Free Links

V1. PPPP. **Proper Public Planning Principles:** 'Engineering Society', Responsibly

SLIDES = http://concepts.gilb.com/dl980 (pdf, 230620 VERSION). Origin of much of this book.

Video (90 min.BCS Lecture, 23 June 2020) = https://youtu.be/mIaVLHvQOp0

V2. SP. **Sustainability Planning**

https://tinyurl.com/UNGoalsGilbVideo

V3. SA. **Sustainability and AI.** Video Podcast 24 mins., Oslo 2019 Aim

https://www.youtube.com/watch?v=J70zf1gF2b8

V4. **Technoscopes** BCS SPA 2020

https://www.youtube.com/watch?v=920rCFYW3ZQ&list=PLKBhokJ0qd3_wlvr0j85YhmNfNj8ZJ8M-&index=2&t=0s

V5. VA. **Value Agile** Video. https://lnkd.in/dkyJpMZ

V6. VR. **Value Requirements** video 22 April 2020, 3 hours.

https://www.youtube.com/watch?v=ZHrwQtG6IMw&list=PLKBhokJ0qd3_wlvr0j85YhmNfNj8ZJ8M-

V7. VD. Video **Value Design**, May 2020,

https://www.youtube.com/watch?v=y_FaiH5jt6E&list=PLKBhokJ0qd3_wlvr0j85YhmNfNj8ZJ8M-&index=4&t=0s

V8. VM. **Value Management** 2.5 hours, 13 May 2020, BCS https://www.youtube.com/watch?v=mr9gUFWj4Jg

V9. QQ. **Quantify the un-quanti fiable: Tom Gilb at TEDxTrondheim** **17 minutes**.

V10. **Generic** Gilb Videos. Search browser for 'Tom Gilb Videos', and hit the 'Videos' selection too.

V110.1. **gilb.com** has a large selection of videos, free and paid courses. https://www.gilb.com/blog?tag=video

V12. **In Projects, why do Managers Bullshit about their Critical Values?**, https://youtu.be/fFWpxrwvPw8 , 42 mins

V13. 2019 WUD Keynote, **"DOOMSDAY: Is the world doomed because we cannot express our Sustainability and AI Goals clearly?"**, slides: http://concepts.gilb.com/dl964, 23Nov 2019, #WUDSilesia, VIDEO= **https://youtu.be/BUXVJgWJSMI**

V14. **Tom Gilb: 10 Suggested Principles for Human Factors Systems Engineering,** lecture from WUD Silesia conference 42 m, https://youtu.be/TlDCwmVgDJQ, [S11]

R Other References

R.Intel: R1. INTEL 2011 AND 2013. Practical industrial cases. SQC and Planguage

https://selab.fbk.eu/re11_download/industry/Terzakis.pdf (Slides and experiences)

R.ValPlan: R2. VALPLAN INFO
https://www.gilb.com/valplan, actual app is at ValPlan.net
I should declare a personal interest in this company.
1. Based on my ideas [B1]. 2. Our company is marketing it.
Richard Smith, UK, is our developer/designer hero building it.

R.GraphMetrix: R3. GraphMetrix.com
I should declare a personal interest in this company.
(Advisory Board, Investor, Using my Ideas [B1].)

R.Eggplant: David Chapman, https://meaningness.com/eggplant/rationality
I know there is something here related to my ideas, but I am just figuring it out.
150820. #problemsolving #metarationality. In any case deep mind-blowing ideas.

R.LandingZones: Erik Simmons, BEST PRACTICES WHITE PAPER , Landing Zones, Available by email request from Construx. CONSTRUX.COM 2020, Version 10.1.3, August 2020, See also: http://wirfs-brock.com/blog/2011/07/20/introducing-landing-zones/

End of Book References

Concept Glossary

Aligned: Enterprise Architecture is synchronized with all significant external and internal Enterprise forces, and plans: updated, precise, supporting, relevant, non-conflicting, transparent, and future oriented.

Ambition Level: an initial informal statement, from a stakeholder about the degree of a value improvement. Needs to be translated into clear and structured Value Requirement specifications.

Architecture: a design process, producing a specification (The Architecture Spec) which is the top-level design process from a defined point of view, and which co-ordinates, or balances, all subsidary considerations of value, resources and constraints.

Attribute: a characteristic of something. A quality, a cost, a function, anything which can describe and distinguish one artifact from another.

Attack, which is a *successful* penetration of a Threat into a system. The Threat has 'materialized' in practice

Background: planning specification which is not the core set of ideas, but is intended to give additional context for the ultimate purpose of prioritization, risk management, quality control, and presentation.

Backroom: the place where design ideas are readied for implementation.

Benchmark: a class of reference level on a Scale of measure. It includes Past, Status, Ideal, Trend. It is used as Background specification to allow us to compare with Targets and Constraints.

Budget: a constraint level for a resource requirement.

Constraint: a requirement intended to restrict, to stop, to hinder us with regard to other requirements, possible designs, and any actions.

Damage:Damage is the negative consequences of successful Attacks to a system

Decomposition:(+030920) refers to a process of decomposing into more-detailed sub-components, such as Sub-architectures, and Sub-Values, any architecture specification objects (including functions, values, resources, architecture, constraints, time) so as to obtain smaller specification-objects, for any purpose; such as early delivery, optimization, separation of concerns (like suppliers), & managing risks. See Part 4 here, and [B2.**Decomposition**: https://tinyurl.com/VPDecomposition].

Defect: a Specification Defect is a violation of official specification Rules. It is poor practice and can lead to problems of using the specification correctly, and timely.

Design Idea: (noun): any specification which is *intended* to help satisfy a higher level of Value, Cost and constraints.

Design (verb): the process of identifying and evaluating Design Ideas, for the purpose of satisfying stakeholder values within constraints imposed.

Design Component: any part of a larger design set, or architecture, which has some notion of independence. For example that it can be implemented incrementally. It can be removed or replaced.

Design Constraint: A requirement specification, that demands or forbids something regarding a design.

Design To Cost (D->C): a well-established engineering concept. You can find designs to meet a given cost requirement.

Design To Value (D->V): the same concept as Design to Cost, except the design process is directed towards meeting a Value (including any quality) Requirement Level of Performance.

Downstream, Upstream: downstream refers to a process to be carried out at a later stage. Upstream, a previous process.

Dynamic Design to Requirements (DD->R): A Cyclical Design process, to meet any set of Value and or Cost requirements, but using measurement, after incremental design-implementation, comparing with requirements, predicting future cost and value levels, and re-designing, if necessary, to better reach the requirements. Note The Planguage Evo method (CE, PoSEM), and the IBM Cleanroom(2) Method both do DD>R. A term coined by Tom Gilb

Efficiency Architecture: This is the ratio of a set of architecture values over a set of architecture costs. (added 020920 TG, see paper 2.)

Entry Process: a simple short QC process proceeding any main process, where Entry Conditions, of any useful kind, are checked as a prerequisite for proceeding to the main process. The intent is to make sure we do not waste time or encounter failure in the main process. The cost of the Entry Process should be very small compared to the average results if we did not use it. Above all we use to to motivate people to take the Entry Conditions seriously.

Environment: implicit, the critical design requirement stakeholder environment. An areas or scope where can can and must expect to find critical design requirements, if we study the stakeholders there and their needs.

Exit Process: a Quality Control (QC) process after any Main Process to try to make sure that it is well done and the outputs are good enough for downstream use. A number of tailored-for-process Exit Conditions are checked and if all are satisfied, Exit is permitted. If any one Condition fails, no exit is permitted.

Frontroom: the place where design ideas are actually incrementally integrated into real systems.

Function: an action, do something, a description of what any system *does*. It contains no hint of information about the other attributes of that function, or its container system. Nor any hint of the designs used to create those attributes for the function, or the system.

Icon (Plicon): a graphic symbol which is assigned a Planguage concept. There are two topics, a drawn icon, and a keyed icon. The purpose of icons is to create a human-language independent symbol like music notation, or electrical notation.

Ideal: a perfect level on a Scale, such as 100% availability. Usually not attainable in practice, or without infinite costs.

Implementation Responsible: a person (or group) which has taken named specified in the spec object, responsibility for actual practical implementation of a design object. This can be for a requirement level (reach the requirement Goal), or for a design (deliver the design and try to get the maximum value from it).

Meter: a parameter which sketches major elements of a measurement process, for a particular Scalar Value or Cost.

Mitigation: Mitigation, is any Strategy which is intended to deal with a Threat (potential Attack), or an actual Attack, within the system, or by being added on after the Threat has successfully penetrated the system (Attack has occurred). Classes are Built-In Mitigation, Post-Damage Mitigation, Planned Mitigation, Threat Mitigation, Attack Mitigation, Post-Attack Mitigation. [P18, and P4:CG]

Open-Ended Architecture: any architecture devices which make it easier to change the system through time.

Owner: a Specification Owner, parameter name shortened to Owner, has the exclusive right and responsibility for updating a given Specification Object, such as a requirement.

Parameter: a Planguage-defined Term, which announces the specification of its defined type of information, about a Specification Object, such as a Value Requirement.

Past: a Scale level which is historic. We can usually document in the Past statement, when, where, who etc. Any useful set of Scale Parameter attributes.

Performance: a systems engineering classification for the set of Value attributes. They include all qualities, speeds, work capacity, savings and any other positive attributes valued by stakeholders.

Planguage: a Planning Language invented, developed over decades, published in many books (from 1976 Software Metrics, Data Engineering, perhaps earlier books), and papers, by Tom Gilb, with feedback, maintenance, and creative improvements from Kai Gilb and many other professional collaborators. It is a systems engineering language, with focus on Values and Costs as primary drivers.

Policy: A policy is a set of principles for decision-making, which permit delegation of decision-making to other people, at other times, under 'unknown conditions at the time of writing the policy'. However, policy may be ignored for higher priority considerations. For example, because of a law or contract in conflict with the policy.

Principle: A principle is a short basic statement, which summarizes and teaches basic philosophy or the pragmatics of a method.

Prioritize: to decide sequence of activation.

Procedure: a specified sequence of activities for a defined purpose.

Process: a continuous, repetitive procedure with a possible ending when complete.

Quality: How Well a function functions. Often ending in '-ility'

Requirement: a stakeholder-desired future system state, which can be tested for presence, or measured for degree: but which might be impossible to deliver in practice.

Resource: any attribute which might be consumed, might be limited, and might be needed to build or maintain a system. Money, time, people, dominate, but many other resource concepts are potentially useful, such as image, qualities, functionality, space.

Risk: a risk is something that can go wrong. An 'opportunity' is by contrast, something that can 'go right', get better. Risk is the possibility of Damage occurring. The **degree** of Risk is determined as a combination of an Attack and a Mitigation.The higher the probability (frequency) of Attack, the higher the Damage. The higher the probability that Mitigation succeeds against the Attack, the lower the total Damage.

Rules: a standard in Planguage which specified the recommended way to do, or not do, a specification of any kind. Failure to follow a rules is classified as a specification defect.

Scale (of Measure): a Parameter which defines a Value or Cost scale of measure, for reuse and reference when specifying Benchmarks, Scalar Constraints, and Targets. It does NOT specify a measurement process, that is for the Meter or Test parameter

Scale Parameter: a dimension, announced in [Square Brackets] in the middle of a Scale specification. It is defined using a {set of Conditions}. This device permits quite detailed Modelling of a system, and allows decomposition of problems so that ctitical Conditions can be prioritized. Example: [Sex]

Scale Parameter Conditions: a set of named conditions which belong to a defined Scale Parameter. Example [Sex] = {Male, Female, Other, Unspecified, Unknown, Multiple}.

Source: the named origin: a person, group, stakeholder, document, or URL of some immediately-previous specifications in a Parameter Specification. The purpose is to enable QC, give credibility, lend authority.

Spec, Specification: a written planning item in Planguage: Requirements, Designs, Analysis, Project Plans, presentations.

Specification Object: a set of Planguage Parameter statements, comprising a meaningful unit of informations, typically a requirement, a design, or sets of these.

Specification Owner: a person (or group) which has undertaken responsibility, by name, for the update and maintenance of a specification object, such as a requirement, a design, or a table.

Stakeholder: an entity; human, organizational, or document, from which we can derive needs, demands, resource limits, constraints, and any form of information, which can be acknowledged as our potential project requirements, and specified formally and clearly as a requirement. A 'requirement source'.

Status: a numeric update of the incremental progress of a Scale Level as we incremental deliver a system design components and measure progress towards our requirement levels.

Standards: best accepted practices for developing and maintaining systems. These include, Rules, Procedures, Exit Levels, Concept Definitions, Templates, Scales of measures, and even App conventions.

Sub-architecture: (+030920 SEA) is any architecture component, which has been decomposed from any larger architecture specification. Sub-architectures are particularly useful in agile architecture, where we need small frequent deliveries, and feedback from architectures. We can also prioritize the most efficient architectures (Part 2 Architecture Efficiency)

Target: a level of Value that we are aiming to reach. It includes Wish, Goal, Stretch.

Threats: A 'threat' is something that can *potentially* cause some degree of project failure, lack of success or negative consequences. It is distinguished from an Attack, which is a *successful* penetration of the Threat into a system. The Threat has 'materialized' in practice. It is distinguished from a Risk, which is a result of the combined effect of a Threat/Attack and the corresponding defenses (Mitigation)

Trend: a Background Benchmark level, which estimates the future of that level. Useful for pointing our Value degradation, or potential competitor future levels of Performance.

Use Case: a written graphic description of how a system element might be used in practice. In Planguage it can be covered by using an appropriate Scale Parameter. Example: [Uses] : {Register, Delete, Update}.

User: a person who personally and physically interacts with a system.

User Story: a requirement statement in the format: Stakeholder + Requirement + Justification. This is roughly at the level of an Ambition Level, and can replace Ambition Level as a starting point for formulating a more detailed Planguage requirement.

ValPlan: ValPlan.net is the URL of an App released for sale May 2019 by Gilb International AS. It is based on Planguage and the Competitive Engineering book.

Value: value is perceived stakeholder

Value Analyst: analyzes stakeholder needs, and priorities, and selects critical, or possibly critical, needs and specified them as requirements, at least at the 'Wish' level (potential Goal requirement).

Value Architect: A person or team, who sits at the Apex of the system, and synchronizes all ongoing efforts in order to get maximum necessary value for available resources. Manages to top critical values, and the top level design architecture.

Value Contracting: this contracting can be done at both internal levels and external suppliers, and basically motivate suppliers to deliver value and get rewarded for it.

Value Design Builders: people and organizations who build, physically, logically or organizationally, any design component or related activity.

Value Designer: a generic (all possible design areas) designer (or team) who undertakes to identify possible design components to reach a Value Requirement level, on time. To research them as to all side-effects and costs, documenting such facts in the design object and corresponding Value Tables. The Value Designer might hand over exploration of a design idea to a Specialist.

Value Director: the person, or group, responsible for focusing on the Value Delivery, and reporting to a steering committee or Board about the plans and accomplishments to date in Value Delivery.

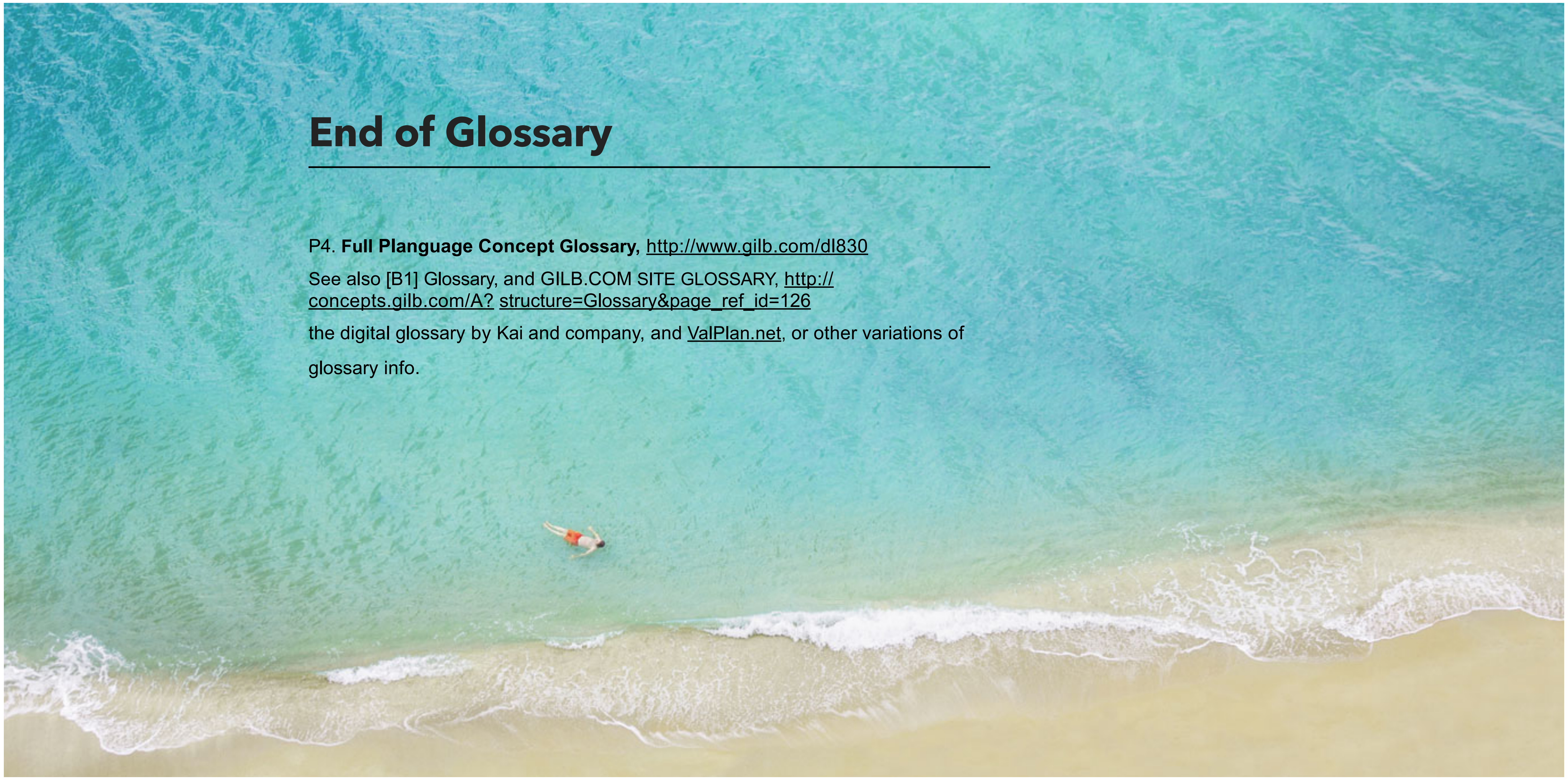
Value Engineering Specialist: a designer with a narrow speciality (usability, security, performance, organizational improvement, AI) who is updated on the state of the art, and has a good international network of people and sources to find good specialist designs.

Value Policy: this is the written policy that gives clear guidance to the Value process, from organizational management. Perhaps Chief Technical Officer level.

Value Process Manager: a person or team responsible for getting a best possible value stream flowing from the other people involved. Sort of like old project manager, except they are focussed on the Values/Costs numbers, not building stuff. They allocate resources (money, time), and assign people to specialist tasks.

Value Quality Control: these people carry out Specification Quality Control of specifications, to make sure the Defects Per Page is economically low enough before Exit to any other process. They are also responsible for measuring value levels and costs after incremental implementation. They will check that designs are in fact implemented as specified by suppliers for Exiting to integration delivery.

Value Suppliers/Sub-contractors: internal and external to the organization people or organizations who undertake a specific responsibility, for construction, implementation, organizing, designing, or any other activity needed to deliver value.

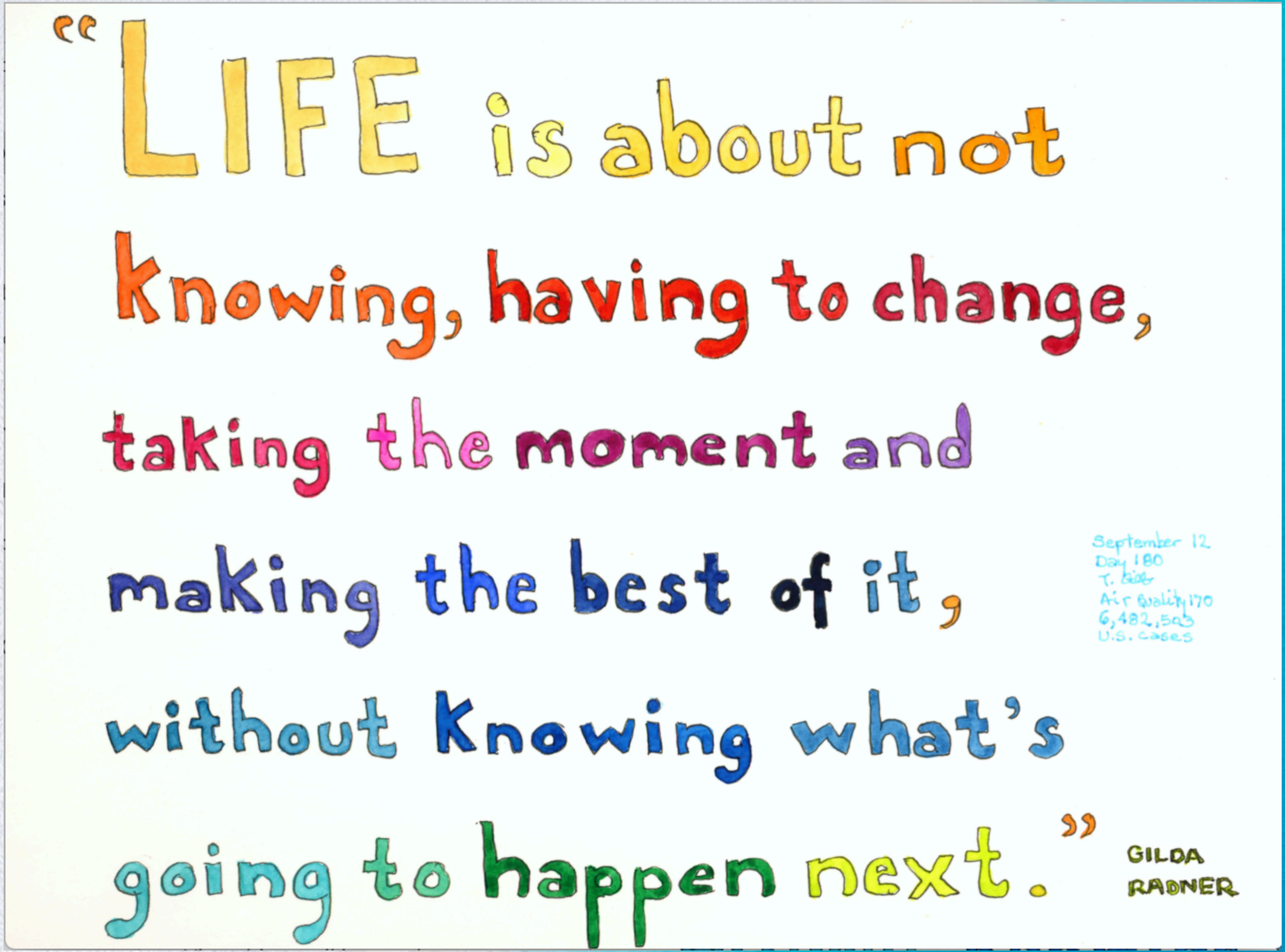


End of Glossary

P4. **Full Planguage Concept Glossary,** http://www.gilb.com/dl830

See also [B1] Glossary, and GILB.COM SITE GLOSSARY, http://concepts.gilb.com/A? structure=Glossary&page_ref_id=126

the digital glossary by Kai and company, and ValPlan.net, or other variations of glossary info.



"LIFE is about not knowing, having to change, taking the moment and making the best of it, without knowing what's going to happen next." GILDA RADNER

September 12
Day 180
T. [illegible]
Air Quality 170
6,482,503
U.S. cases

Figure 42: Source Drawing Tyra Gilb (Sister, California), and Quote: Gilda Radner (SNL, -1989)