Tom Gilb

Value Requirements

Figure Cover, **Source: "To Catch a Butterfly: Epistimic Miracles of Serendipity. The.xel.io**

http://te.xel.io/posts/2018-03-04-to-catch-a-butterfly-epistemic-miracles-of-serendipity.html

Introduction

This book will help you analyze and specify the most important requirements for your project, how to quantify success 'values', so you can manage them, and how to structure them, so as to reflect your complex reality.

This book will help you with the following work processes: stakeholder analysis, value requirement specification, requirement quality control, requirement prioritization, risk management, clear communication, and systems-level thinking.

It will help you set the stage for design, estimation, contracting, and project management.

The method is based on our advanced planning language, 'Planguage', which specializes in values, qualities and costs like no other alternative requirements method.

Simply the best, for those who must succeed, and cannot afford to fail to deliver real value.

Chapter 1. What are `Value Requirements` ? (VR)

Value Requirements are the most important requirements for any project. They are the main purpose, and main justification, for a project. They are the *stakeholder's* values.

Value requirements start life as value 'attributes' *needed* by 'stakeholders'. No project can deliver all 'needed' values, by a deadline. No project will find all stakeholder values to be *worth* delivering.

So all value requirements start life by being *acknowledged* as *possible* delivery candidates. But VRs need to go through an *evaluation process* to determine that we can prioritize them for real delivery.

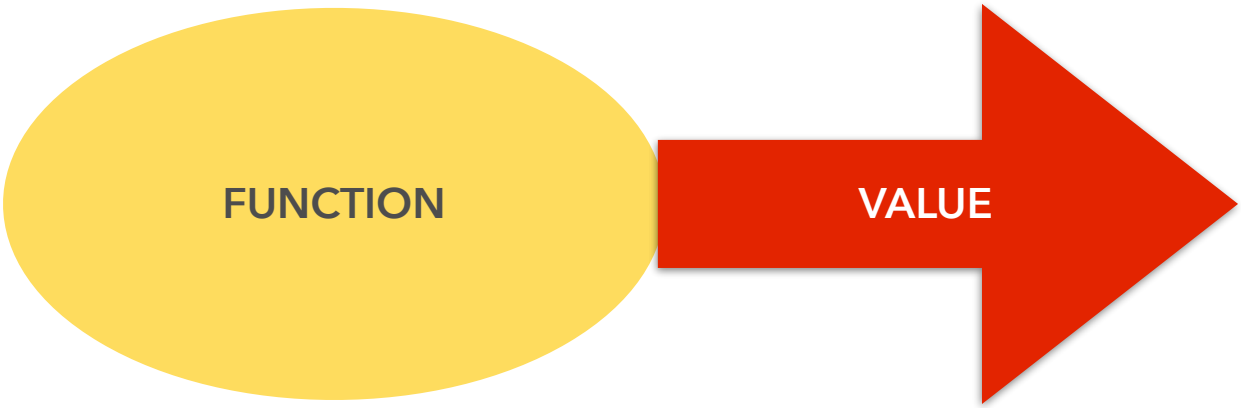


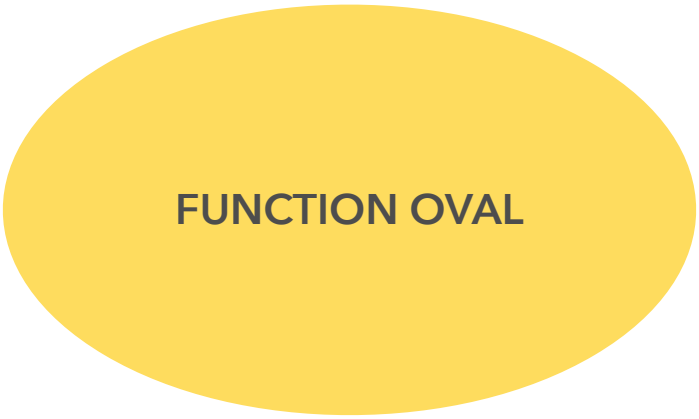
Figure 1.1 : A Value is a variable level of performance for a function. Represented graphically as an arrow emanating from a Function symbol. This is a simplified model, with a *single* value arrow. Reality is always that *multiple* values are needed concurrently.

What *other* kinds of requirements do we need to consider?

There are several other 'requirement types' which you will need to consider, but we will *not* treat these in detail here. They are treated elsewhere (Competitive Engineering, CE).

Here is a list of other (not 'value') requirement types.

Function Requirements: WHAT the system must DO.



The Function 'keyed icon' is: (any oval keyboard symbol) '**O**'

Figure 1.2 : a system function, represented as an oval shape in Planguage icons.

Planguage icons are a formally defined set of symbols, like music notes, or maths symbols, which represent systems engineering concepts[1]. And which are independent of human spoken languages.

🎶 ❤️ ♾ 🏴‍☠️

[1] http://www.gilb.com/DL386, Full Planguage Concept Glossary. Including Icons.

Resource Requirements: limitations on any kind of resources (people, time, money).

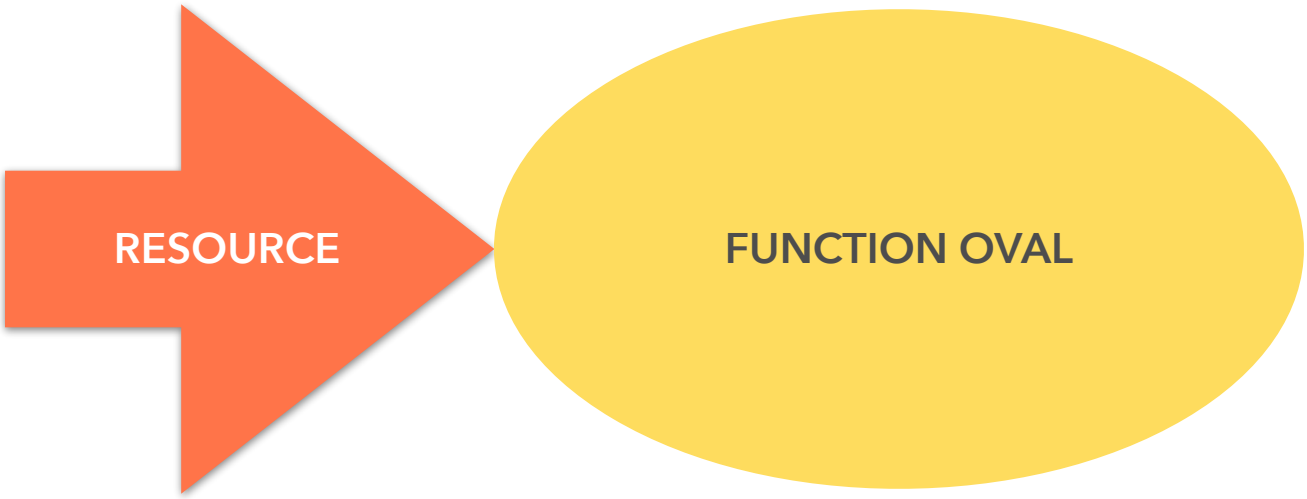


Figure 1.3 : a single resource arrow giving a function the potential of some values.

The 'Resource' 'keyed icon' is: **->O**

Binary Constraints: legal constraints, design constraints, anything that must either be **done**, or **not done**.

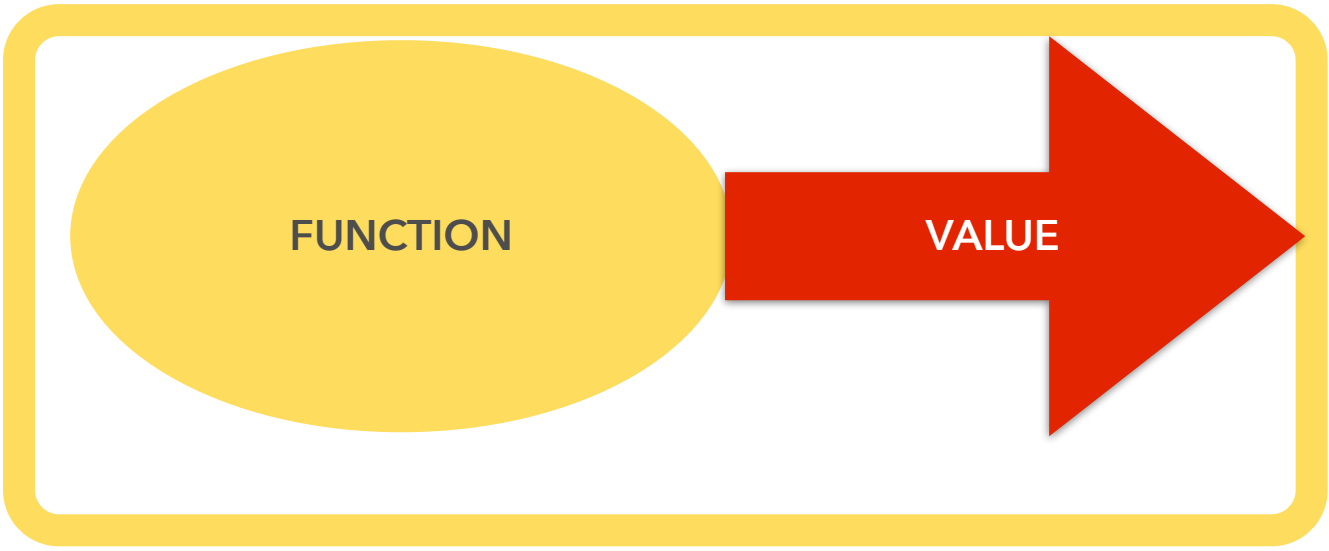


Figure 1.4 : A constraint (the rectangular shape) which the system must be 'within'.

Another type of constraint a 'Scalar Constraint' will be dealt with in this book. It is a numeric limit (not too hot, not too cold) on a Value Scale (at least 99.99% Availability) or a Resource Scale (finished absolutely latest ny end of year, no matter what).

Part of the reason we are *not* treating these non-value requirements in detail here is:

1. They are fairly well understood.

2. They are comparatively *simple*, compared to Value Requirements.

3. We want to focus on the less-understood, but extremely-decisive, Value Requirements. The main point of all projects.

What kinds of 'requirements' are NOT on our 'Value Requirements' agenda ?

1. User Stories (see Chapter 15.1)

2. Use Cases (See Chapter 15.2)

3. Simple written text (like 'better safety')

4. Designs *claiming* to be Requirements

Can User Stories be used?

User stories do not contain enough information to serve as value requirements. But you can add on the sort of information we present in this book, to make them serve as value requirements (ref.

A, US)

User Stories have a useful structure:

1. A stakeholder (narrowly called 'user')

2. A 'story', which functions as a sort of requirement. But is not detailed enough for serious purposes.

3. A justification for the story, which is good practice. But can be improved upon.

My initial advice for people who have to deal with User Stories is to write them up as a Planguage statement, and then proceed to derive more-useful detail from it.

For example:

Usability:

Type: Value Requirement.

User Story: As an expert user I want shortcuts to save me time. <-US030719.

Scale: Average cycle time in minutes for a [Task] by a [User].

Pro Level: **Wish**: 6 minutes, Deadline = End Next Year, Task = Expert Complex Tasks, User = Expert.

Comment: in translating the user story we have carefully avoided the 'shortcuts' which is an amateur 'design' suggestion. We have focused on the stakeholder value of saving time, and left the detailed design, to achieve that end, to a professional UX designer.

Specification example 1A: the user story is cited, then translated into a value requirement (Scale and Wish statements). The 'scale parameters' [Task] and [User] are used to make a more general 'Usability' specification than the 'expert user' in the user story, and to specify a wider range of tasks than the unspecified tasks in the user story. The result is that we can specify a wide variety of Usability value requirements.

We can for example add a statement to the Usability specification *above* like:

Beginner Requirement: **Wish**: 10 minutes, Deadline = Beginning Next Year, Task = Beginner Frequent Tasks, User = Beginner.

Specification example 1B: We added a second 'Wish' value requirement, to the Usability specification above. This has several advantages. We can now prioritize one of them, based on value and cost, and deliver the value *early*; without waiting for the other Wish to be completed.

Chapter 2. More detail on the nature of Value Requirements (VR)

- VRs are often qualities, '-ilities' like reliability, security, usability: 'How Well' the function performs. Often called 'performance attributes'.

VALUE SPECIFICATION TYPES

Specifications			
	Requirements: Future Needs		
		Value Requirements: How Good	
			Qualities: How Well
			Other Values: How Much
		Functions	
		Constraints	

Figure 1.5 Value Specification Types.

- VRs are, in systems engineering, classed as *Performance* attributes. 'How good' the function is. 'How Good' includes: 'how **well'** (Qualities): but also 'how **much**'. For example speed,

volume, frequency, sales, market share and savings. Values which we would not call 'qualities'

- VRs are always a 'degree of system performance' which is 'actually *valued* by some stakeholder'. If not valued, then by definition, it is of no worth to *any* stakeholder.

- VRs are always defined as a desired 'numeric range' or a '*point*', on a 'scale of measure', which means the value is a *numeric* value.

- In addition to a value requirement being a numeric level, that level must *also* be achieved by defined *times* and *conditions*, for the total requirement to be fulfilled.

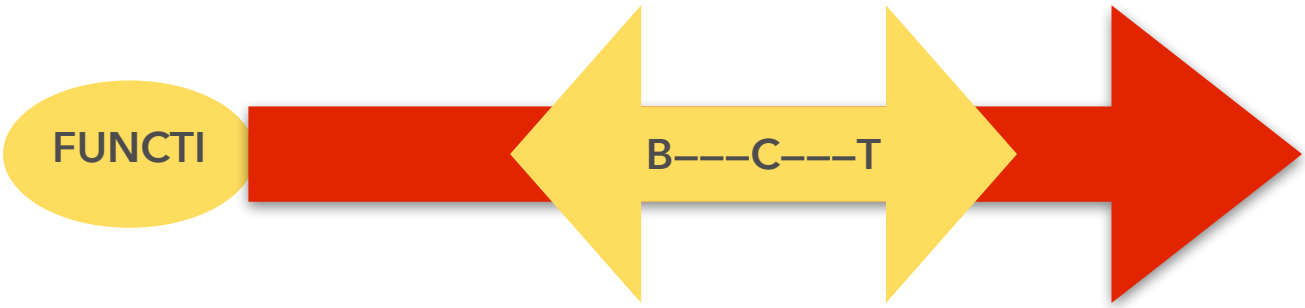


Figure 1.6 : on the Value Scale (Red Arrow symbol) a value requirement might be expressed by numeric constraints (C) and by numeric Targets (T). In addition a Benchmark level (B) might be added to a requirement specification to inform us of past and current levels of performance, for comparison with the required levels.

In our planning language, Planguage, this might be expressed like this.

Security:

Scale: % probability of detecting a hacker within 5 seconds.

Status: 10% last year. (Benchmark level)

Tolerable: 80% by End this year. (Constraint Level)

Wish: 98% by End Next Year. (Target Level)

Example 1C: a value specification.

Security is the reference tag for the entire specification.

Scale is a parameter in Planguage for defining a value variable, such as Security, so that the various levels of Security can be expressed numerically.

Status gives us the moving current change of status in the level.

Tolerable gives us the bare minimum level which is acceptable.

Wish is the stakeholder-desired, or stakeholder -needed, level of Security, on that Scale.

Chapter 3. The Scale In more detail.

3.1 The essence of a 'Scale specification'. And a 'motivation' to use it.

The most powerful, and useful, requirements method-detail that this book can inform you about is the 'Scale'.

This is because your 'Scale specification' moves you away from informal and fuzzy requirement specifications, and over to clear, logical, quantified methods of thinking about problems.

A Scale specification means you are moving your entire approach to projects from primitive and failure-prone communication with others, over to 'engineering' and 'science': over to a fact-based culture, to an evidence-based culture.

This takes a little more effort than, 'being lazy, and failing in your projects', but I assume you are reading this book because you want the skills to improve your capability and success, in your profession.

Another thing about the skill of 'defining values in terms of a Scale', is that you can use this skill for the rest of your life; on any kind of problem solving, and any kind of project. It is very good job security, in changing times.

Everything this book teaches is like that: it is based on universal ideas, which are quite independent of current technology, and independent of any profession you might undertake. I know from

60 years of professional experience, where I am still on top of my game; and able to impress top international professionals with these skills.

But this applies not only to me personally: thousands of professionals in major corporations have recognized the benefits

of this skill, and chosen it.

One example is the over 21,000 Intel engineers, over about a 20 year period, who have voluntarily taken a 2-day training course in this way of specifying product values, and practiced their skills.

One simple measured result was 233% overall productivity increase (ref. Terzakis) (ref. A, Intel).

Figure 1.7 Intel

The good news for you personally is that, of the 100% of people who would benefit from these methods, there are probably less

than 1% of them actually using them today. You will be more competent than the 99%, and hopefully you can help spread this culture of clear thinking?

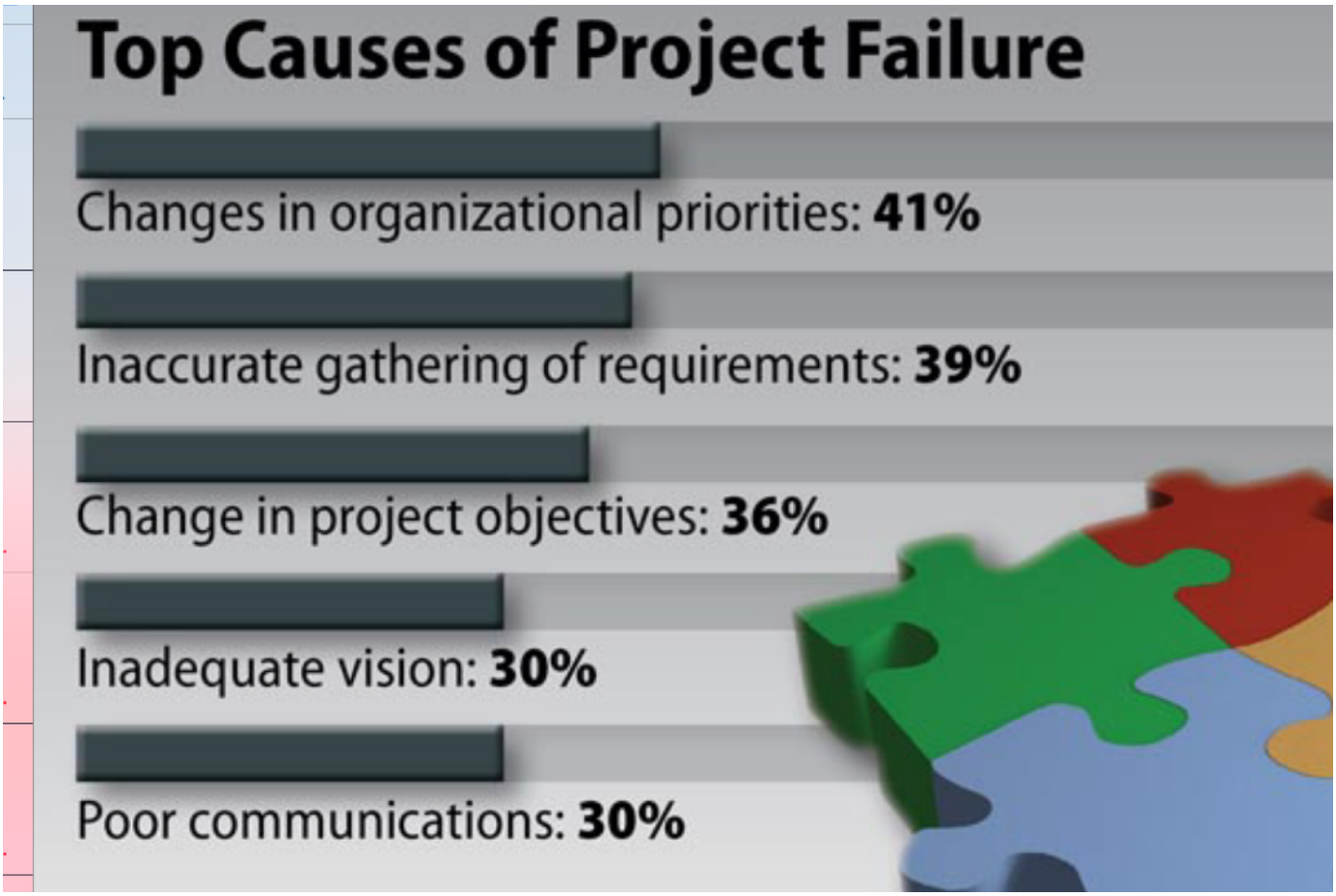


Figure 1.8 : Good value Requirements help us avoid project failure. All 5 of these failure causes are actually related to good 'value requirements'.

Source: http://mobile.baselinemag.com/project-management/slideshows/why-some-companies-have-more-successful-projects.html

Scale definition is the foundation, the core, of this quantified approach to value delivery.

This is true, as many of you are aware, in all sciences, and engineering disciplines. But somehow the business, politics, and

planning disciplines have avoided quantification of many values and qualities, and just used 'nice-sounding words'.

We need to stop these immature practices, for serious projects, in order to improve our success rates.

The 'Scale' parameter specification is used to define success, and to define failure, so that we will know how far we are from success, and how near we are to failure, at all times. And we can take real-time action to succeed, and to avoid failure.

The 'Scale' parameter is not only a clear definition of success and failure, but it is a tool to help us, as teams and groups of people, to communicate success and failure, so that all parties understand these ideas exactly the same; misunderstanding and misinterpretation should be near impossible.

This is important when projects are widespread, geographically, culturally and legally. And when change is the only assured constant.

The 'Scale' specification, is about an idea of *variability*: an idea for quantification: a platform for 'putting numbers on values', so as to express ideas of 'degree, of goodness' or 'badness', or 'improvement', or 'comparison'.

Scale is NOT an idea of 'measurement' (how to determine where you are now on that scale). We leave specification of *measurement* ideas to another parameter, the 'Meter', as in speedometer and voltmeter.

Once a Scale is defined, we reuse it for a very wide variety of purposes.

In addition, we can use more than one different measurement process (defined by Meter specs) for a single Scale. Quick and dirty, or more accurate and credible measurements, for example.

Our graphical symbol for a Scale is an arrow. *Value* Scales emerge from a system's function, and *resource* Scales point into the function - supplying a system with resources to drive the values to emerge.

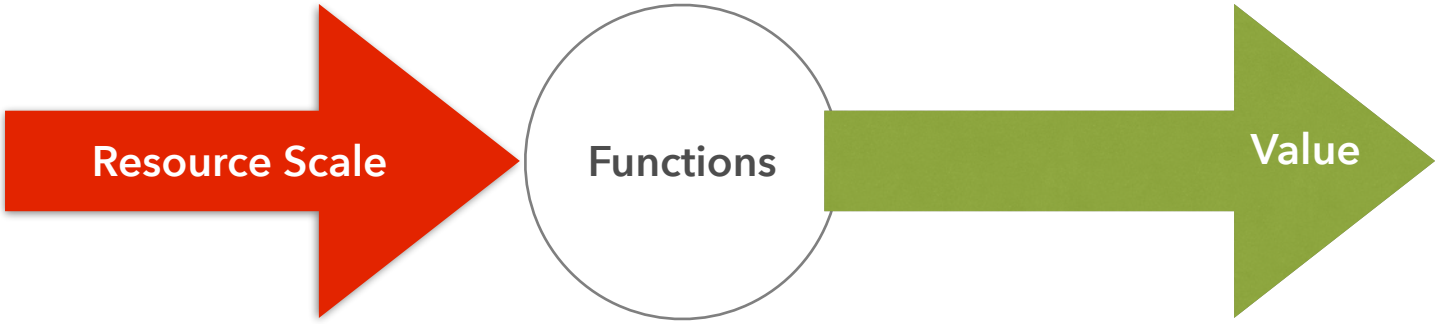


Figure 1.9: Functions (what a system 'does') need a supply of resources (like people, time, money) to produce values.

These *resources* are the prices we pay to develop and maintain 'values', and the 'value levels' we need, *when* we need them.

Initially, functions have no particular values associated with them.

Functions are independent of any particular values.

But for a function to be of use in a real world, it needs some value levels, of things like 'availability', 'usability', and 'work capacity'. If these values are zero or low, then the functionality would not be visible, or useful, in the real world.

If we improve value levels, for your product or service, to certain currently useful levels, we become 'competitive'.

If we improve value levels significantly beyond others in our market, then we can offer superior value to stakeholders, which might command their willingness to choose us, rather than competitors.

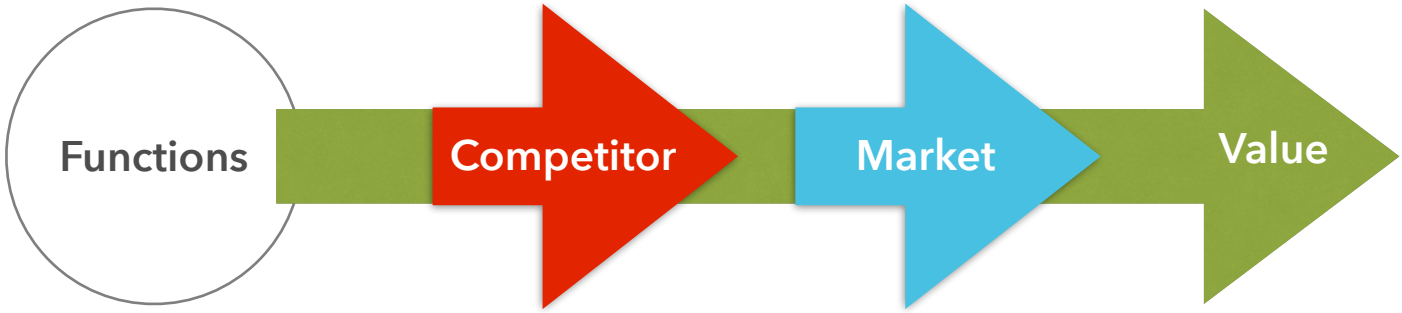


Figure 1.10 : for the same function, the basic market or business, like banking, or plumbing, or Yoga Training, you can plan to deliver a better level of one-or-more values (Market leading levels), to win the business against the competitors.

But this must be a clear idea, well-delivered in practice, and without sufficient immediate competitive response from your best competitors! This is a process for winners only: having clear and useful scales of measure, is a basic minimum tool, for this competitive action. 'Nice words and intentions only', are for losers.

3.2 Minimal Scales.

Here are *the minimum* attributes of a Scale parameter specification:

- Must allow scale numbers to have a 'useful meaning', when associated with the scale,

- Should not be so short a Scale-specification as to leave critical concepts undefined, or ambiguous to any reader.

Here are some desired attributes of any Scale specification:

- It should be intelligible to domain specialists,

- Should be a good reflection of the value, as perceived by the domain specialists, and other relevant stakeholders.

So here are some reasonable, and simple examples: (prefaced by a tag ('Usability', etc.), to give some context)

Usability:

Scale: % of users who can master the basics within first day of use.

Impressiveness:

Scale: % of people who took a test ride, who then joined a waiting list within a week.

Example 1D

I am not saying these are as good as we can make them, but they are ok for many purposes. They are not good enough for complex, large, critical systems. But they beat most non-quantified value requirement statements, like 'very impressive', or 'highly user-friendly'.

And here are some not so good examples:

Security:

Scale: Number of hacks.

Example 1E

Why? There is a failure to say more about 'who' did the hacking, what 'type' of hacks, the 'time period', the 'object hacked'. Just *too many* unspecified things.

Co-operativeness:

Scale: % of acceptances to join.

Example 1F

Why? Too many related conditions not defined here, like 'join what?', What kind of Invitation? Over which time period?

3.4 Developing a Scale from an Ambition Level.

An **Ambition Level** is an informal statement of a requirement, about one sentence long. It often comes from an 'official', attributed source. The problem is that it is filled with ambiguous terms, and does not lend itself to quantification. So it becomes our job to clarify and quantify 'His Masters Voice'.

We could of course complain that the source (our boss?) is sloppy. But that would be unnecessarily undiplomatic.

Instead we should joyfully accept the challenge of articulating what the *power that be,* said. After all, as I say:

He who taps the keyboard holds the real power.

Note that this Ambition translation process is essentially the same as the design of a Scale from a User Story, as explained above in 'Can User Stories be used?'

Here is an example of an ambition level:

Ambition: "before performance, Tesla prefers to focus on safety first' <- Elon Musk. 140319

Example 1G

And here is a Scale we can derive from that:

Scale: % average passenger safety rating by Euro NCAP

Example 1H. https://www.euroncap.com/en/results/tesla/model-3/37573

To derive that Scale we had to think:

- What kind of safety ratings give useful objective proof of safety?
- What units of measure are used in them (stars, % survival) ?
- Which units of measure, if there are several, serves our purposes in this project?
- Some searching on the the internet (Tesla, Safety Ratings) might give specific options of ideas.?
- Would the power-that-be (Musk) think this is a good scale of measure for *his* purposes?
- Is our suggestion broad enough for purpose, or is it unnecessarily narrow?
  - Does it cover all market areas?
  - Does it cover all types of the value (safety)?

The 'level' is missing!

There is a specification element missing in the **Scale** spec, which is 'exactly where on the Scale we are targeting for the future', our Ambition **Level**.

We have (Scale) derived a definition of the Safety value, suitable for applying (safety level) numbers. But we have not yet derived the *required levels* themselves. So we have only done the first half of the job of interpreting the Ambition Level. Let's say we did the 'Ambition definition' part (the Scale); and now we have to turn to specify interesting levels on that Scale.

We could add such a Level specification, and possibly derive it from the Ambition quotation. This is a subject we will look at below. But it might look something like this:

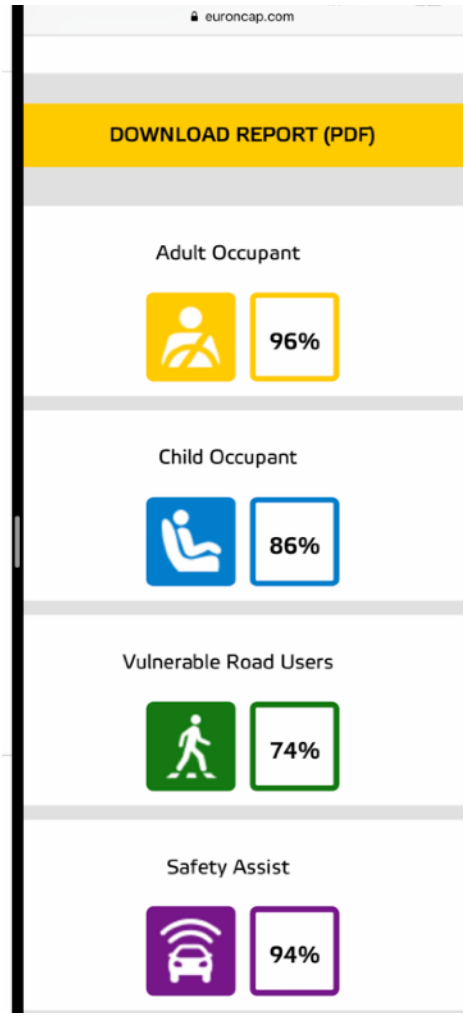
Goal 98% [Within 2 years, for Adult Occupants].

Example 1I

There are several weaknesses with the Scale I suggested above. And I will discuss improvements below. But I wanted to make the point about 'deriving a scale from an Ambition Level' .

This derivation practice can be done in a much more detailed way, when the Ambition level is richer with various concepts (about *when, where, who, what*). We can return to that after the next section on Scale Parameters.

Figure 1.11 : Tesla-3 %-ratings from euroncap.com (URL op cit). We note they use a % scale.

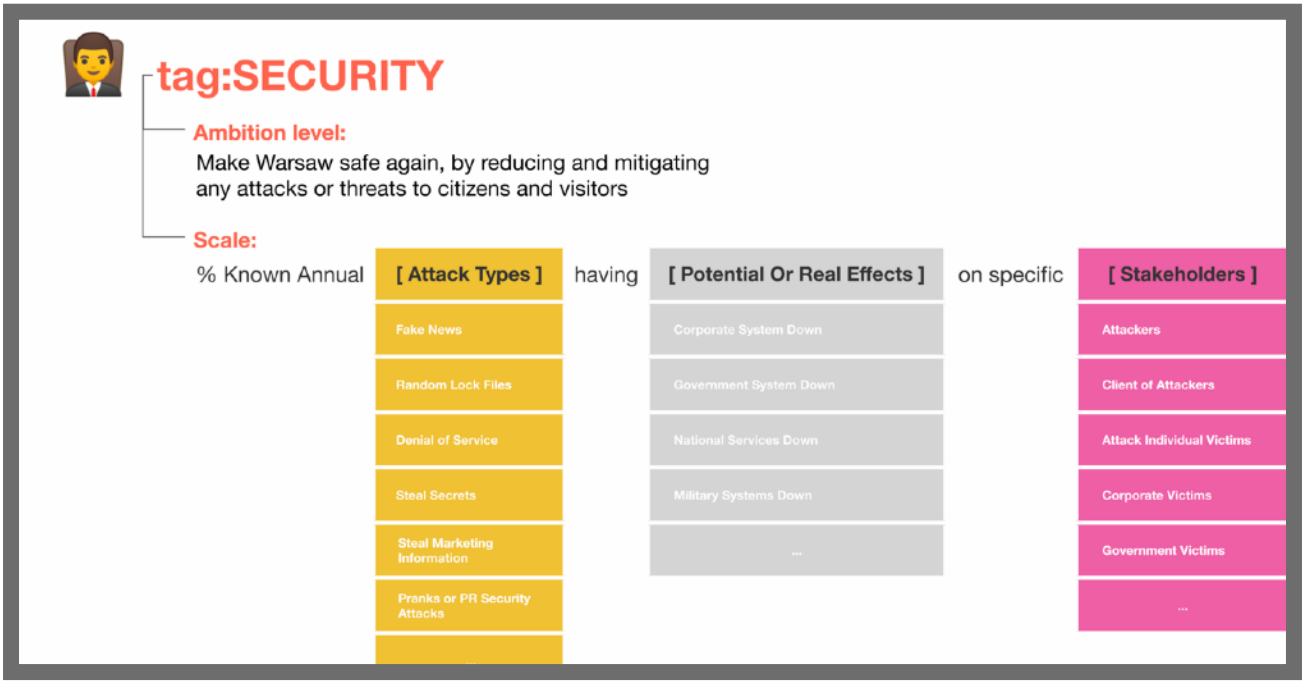


3.5 Scale Parameters

If we want:

- Accurate modeling of large and complex systems
- The possibility of separating critical value-deliveries from less-critical ones, which permits us to 'do critical stuff early'.

- The possibility of much better definitions, so nobody can possibly misunderstand (like contractors and suppliers, and even managers).



Then you will want to improve the 'resolution' of the Scale tool.

Figure 1.12 : Each Value Scale is one of many dimensions of the system's Value Set. Each Value spec can have several [Scale Parameters]. Each [Scale Parameter] can have several attributes which are used as requirement specification'conditions. I might call this Three-Dimensional Value Modeling. Example from May 2019 Master Class, Warsaw planning exercise. Graphic Design Source: anna@Karlowska.PL, 2019

This need for improved 'resolution', using Scale Parameters as a tool, is so common that I find I have to use it on almost every value Scale, on every project, even seemingly small projects.

Here is an example: derived from https://www.euroncap.com/en/results/tesla/model-3/37573

Vehicle Safety:

Scale: Star Rating number for [Person Type] and [Car Specs] for [Safety Equipment] with [Alternative Model Validity] for a [Publication Date] by a [Rating Agency].

Example 1J

Perhaps you can imagine roughly what is happening with this specification. It is intentionally quite readable for a domain specialist (car freak).

The terms in square brackets ([Car Specs]) are:

- Formally defined terms: The **C**apital **L**etters signal that they are defined, somewhere

- General Concepts: defined with a specific set of elements, which as a set, define the general concept.

  - For example: People = {Babies, Children, Adults, Aged}. We are going to sometimes use the *set* '{...}' parenthesis, to list a *set* of things. But sometimes this is not necessary for clarity, so we drop it, for clarity.

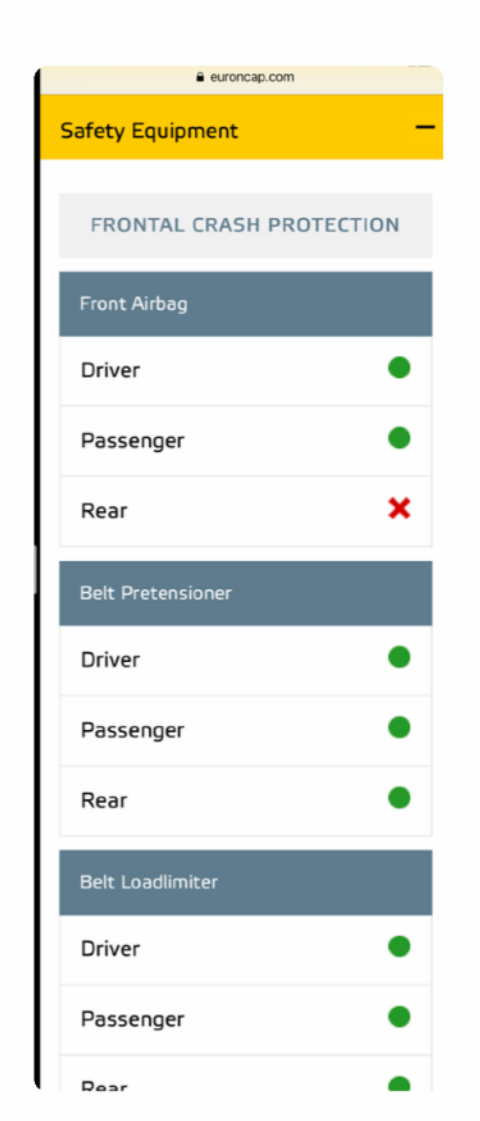


Figure 1.13 : a clip from euroncap.com safety report, URL cited above, which shows the actual structure of the 'Safety Equipment' concept in reality.

Here is an example of making use of the [Scale parameter] structure to articulate a target level requirement.

Wish: 5 Stars, by Next Year, Person Type = All, Car Specs= {Tesla 3, RWD, 4 Door, 2019}, Safety Equipment= {Front Airbag, Belt Pretensioner, Belt Load Limiter, Knee Airbag, Side Head Airbag, ...}, Alternative Model Validity=Dual Motor AWD Model 3, Publication Date=2019, Rating Agency= All.

Example 1K. A 'Wish' level is a stakeholder value target, which is not yet accepted by a project as prioritized, feasible and economic (that is called a 'Goal' level commitment).

Read it slowly, and parse it, decompose it. Or see an edit below.

Here is a more-structured format

Wish:

5 Stars,

by Next Year,

Person Type = All,

Car Specs= {Tesla 3, RWD, 4 Door, 2019},

Safety Equipment= {Front Airbag, Belt Pretensioner, Belt Load Limiter, Knee Airbag, Side Head Airbags, ...},

Alternative Model Validity=Dual Motor AWD Model 3,

Publication Date=2019,

Rating Agency= All.

Example 1L. Same as Ex. 1K, just spread out for readability.

We can specify any useful number of such statements, with any useful valid combination of Scale parameter dimensions we want.

We can home-in on the most critical subsets of Scale parameter dimensions, so that we can focus our energy on, and prioritize, exactly the ones that have the highest value for us, especially in the near term.

This might seem 'complicated' at first sight.

But it is in fact a way of *simplifying* very complex overall problems, by allowing us to carefully extract something simple that we can work on, and deliver some value improvements early, for critical subsets.

Early partial value delivery is about 'learning about complex realities', before we commit to scaling up.

3.6 Defining Other Terms in a Scale

A Scale specification, will use the **Scale Parameters**, to give pretty sufficient *definition* of the Scale Parameter terms.

For example (a definition of a Scale Parameter, in terms of a set of things):

Person Type = Adult Occupant, Child Occupant, Vulnerable Road Users, Safety Assist for Driver

Example 1M: The set of things that make up 'Person Type', serves as a definition.

But there may be *other terms* in a Scale specification which require formal definition in our specification, to avoid ambiguity, misinterpretation (intentional, or not), and consequent problems, delays and costs. But are not defined in terms of a set of things.

It is a necessary defensive practice, a risk-mitigation practice, to formally define these terms somewhere. In the specification, or in project-related glossaries.

The Planguage-agreed signal for formally-defined terms is, as is also the case with Scale Parameters, that we use **C**apital **L**etters in the words of the term, as a signal that a formal definition is available, or should be at some point. When tool support is used, such words will appear as hot link words, one click away from the formal definition.

Example:

Child Occupant: a person under 16 years of age.

Example 1N: Defining a 'Defined Term' with a straight definition; not using a set of things to define it.

My personal practice, when someone asks 'what does that word mean?' is to immediately and always, create a formal definition.

At least to Capitalize the term to indicate my intent to define later.

Merely answering orally is a poor practice, for obvious reasons.

3.7 Scale Libraries

The best scales of measure will be *highly tailored* to the local project environment.

It can however help you, to begin the tailoring process, from *previous* examples.

These Scale definition examples are stored in several ways.

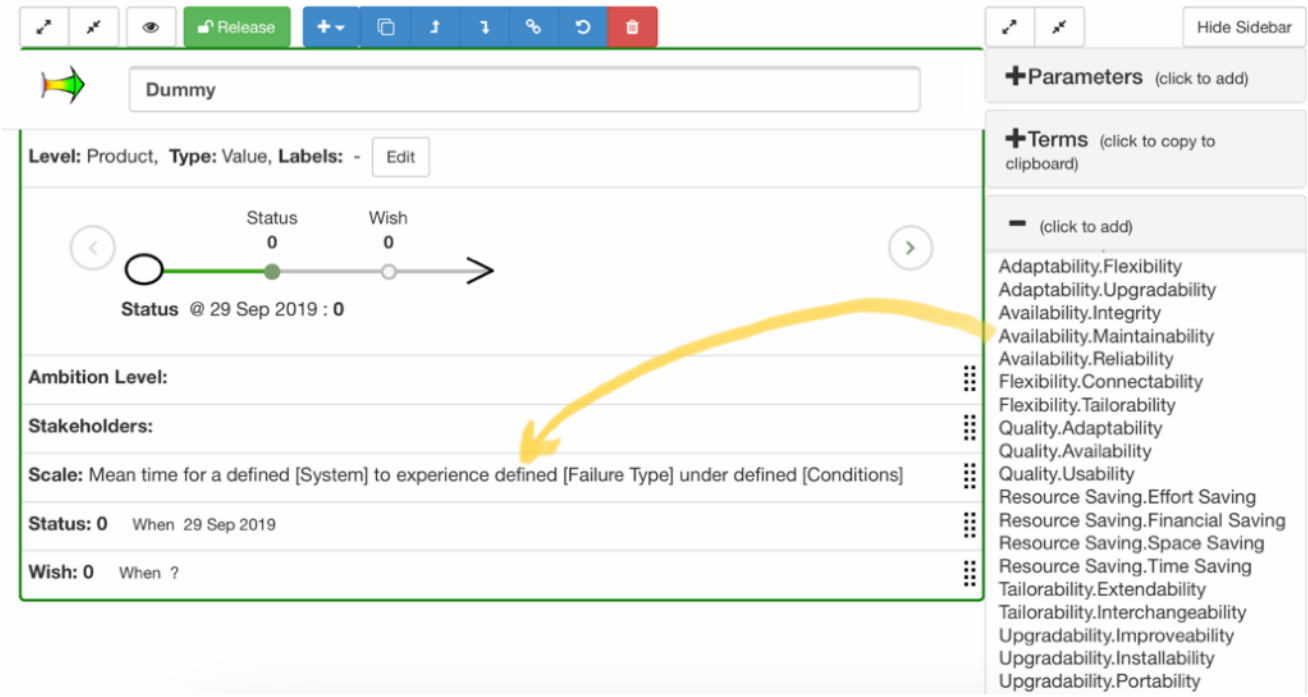
1. Previous projects in the same environment are potentially rich with useful examples of Scales, and the experience of using them in practice (think of the 21,000 Intel engineers and the environment of *chip architecture*).

2. Some very common Scales of measure (examples Usability, Security, Maintainability) are published in books like my Competitive Engineering book: see examples Chapter 5 at http://concepts.gilb.com/Free+Download+Competitive+Engineering+-+Chapter+5.

3. Some of these are digitalized in tools, like ValPlan.net

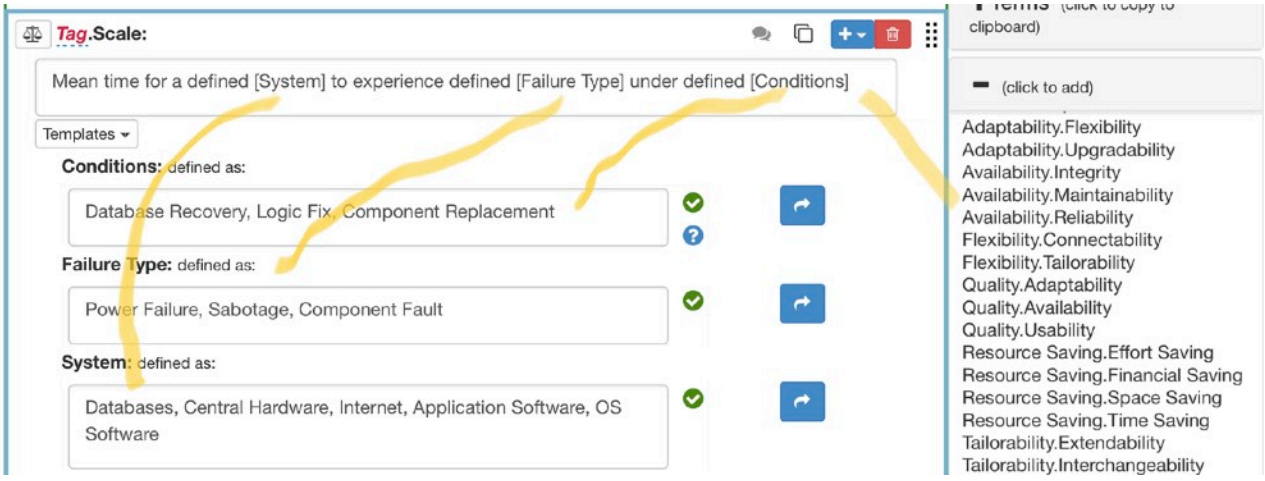
Figure 1.14 : a list of ready-made Scales of measure (ValPlan.net tool). The 'Maintainability' scale of measure was selected from the library of measures[2], and copied into the 'Scale' specification.

It can be modified if desired at this point. But at the least, with it's 3 [Parameters] it is quite general



and can be applied for many detailed dimensions, which are up to us to define in detail. You can continue to add to this Scale library with your own Scales, for reuse later.

Figure 1.15: The [Scale Parameters] from the Scale Library Template can be defined as you wish and



need. For example as above.

---

2 This set of Scales was directly derived from Competitive Engineering Chapter 5. http://www.gilb.com/DL26

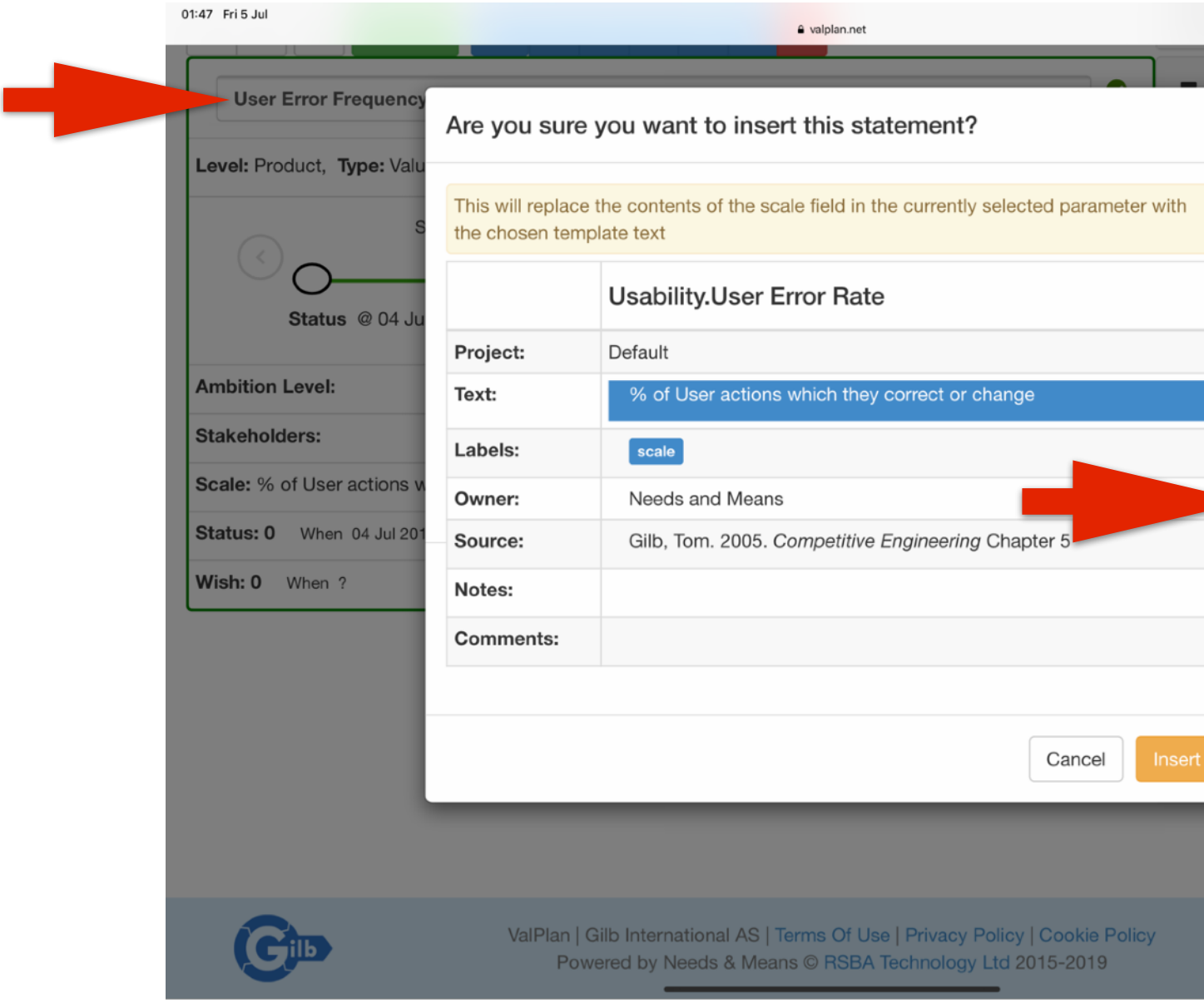


Figure 1.16: In this ValPlan tool example, a Value specification, tagged 'User Error Frequency' needed a Scale of measure.

We looked in the ValPlan library (copied from the Competitive Engineering book) and found a similar value called 'Usability.User Error Rate'.

The Scale looked good enough, so we Inserted it into the User Error Frequency specification window. (Left background, in the Scale).

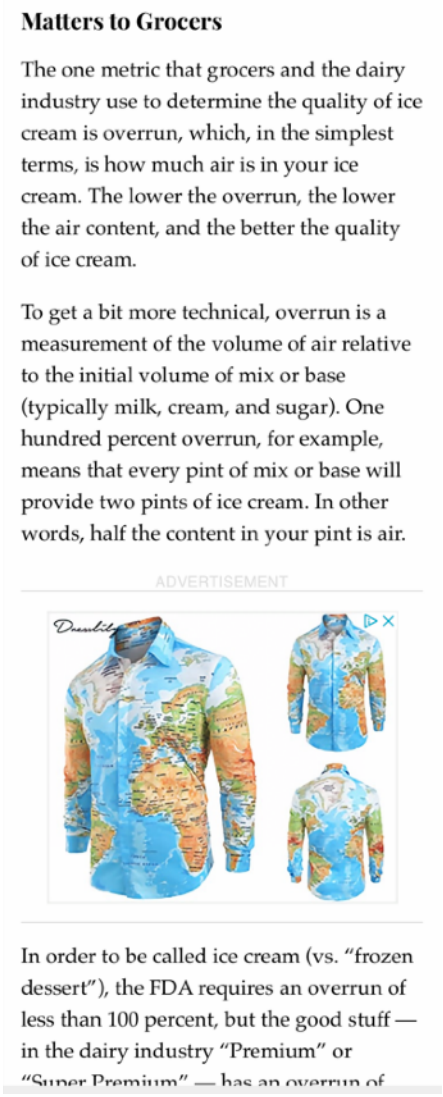
We are now free to modify it to taste. For example by making 'User' a Scale Parameter [User] and then defining classes of User, like {Novice, Advanced, Coach}.

The Internet Library of Scales of Measure.

The internet has a huge collection of defined Scales of measure, for almost anything you can imagine.

Search your favorite Value + the word ‘metrics’

Example ‘ice cream taste metrics’



Matters to Grocers

The one metric that grocers and the dairy industry use to determine the quality of ice cream is overrun, which, in the simplest terms, is how much air is in your ice cream. The lower the overrun, the lower the air content, and the better the quality of ice cream.

To get a bit more technical, overrun is a measurement of the volume of air relative to the initial volume of mix or base (typically milk, cream, and sugar). One hundred percent overrun, for example, means that every pint of mix or base will provide two pints of ice cream. In other words, half the content in your pint is air.

ADVERTISEMENT

In order to be called ice cream (vs. “frozen dessert”), the FDA requires an overrun of less than 100 percent, but the good stuff — in the dairy industry “Premium” or “Super Premium” — has an overrun of

Here is what I found: the first hit,

Figure 1.17 : Quality of Ice Cream Metric

So, before you say

- There is no quantification
- It cannot be quantified
- It is a ‘soft’ value
- I do not know how to write a Scale for this

Search the web for a pretty-good starting-point Scale.

There are always lots of options out there, and you can tailor them for your use.

No excuses. **All** critical values can be quantified, with a defined Scale, easily.

Try *your* interesting value (+ 'metrics') on your phone browser now.

Lots of professionals and academics have struggled with quantification of the same values as you are interested in, and put their experience on the internet. Use it for inspiration.

3.8 Multiple simultaneous Value Requirements

Real problems are not nice to you, they are not simple to understand.

You cannot just focus on a single Value metric and forget about any others.

You are going to have to normally 'juggle' ten or more value metrics at the same time.

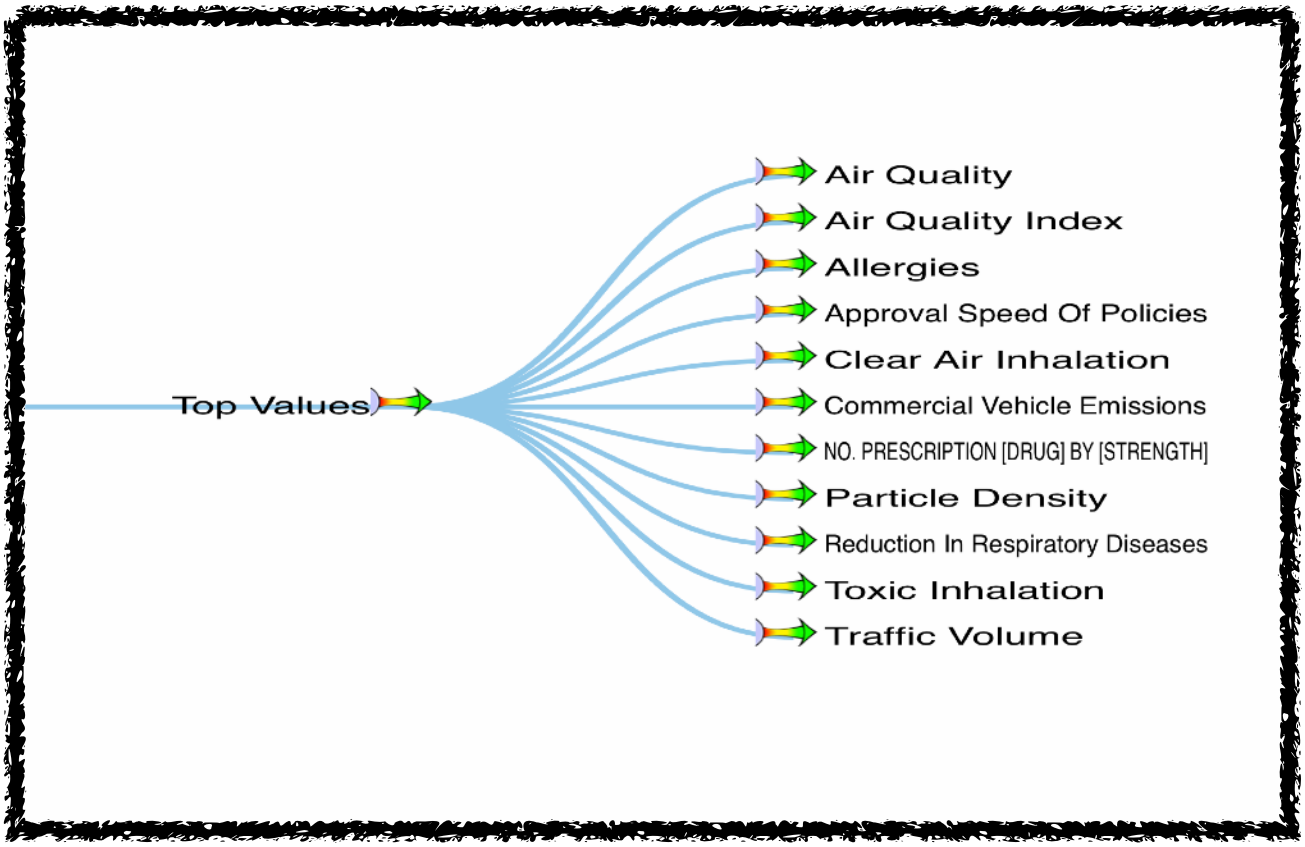
If they all have well-defined Scales of measure: you stand a chance of success.

If they are all in the fuzzy 'Ambition Level' condition: failure is guaranteed, in a stormy sea of confusion.

I recommend that you select a set of the **top-ten most-critical** value-requirements, to focus-on delivering, initially.

Keep all others, lower-priority values, on hold, until you have delivered the first group. (Reference B)

Figure 1.18 : Example of Top 11 Values.



Source BCS Exercise Sept 2017, 'London Congestion for Air Quality'. Notice that this is also a definition of 'Project Value' using the 11 decomposed different values, as the definition-by-subset.

3.9 Here is an example of a single complete Value Requirement Specification, with all the extra supporting detail we will discuss below.

Advance peek a real and complex example: to be explained in this book detail by detail.

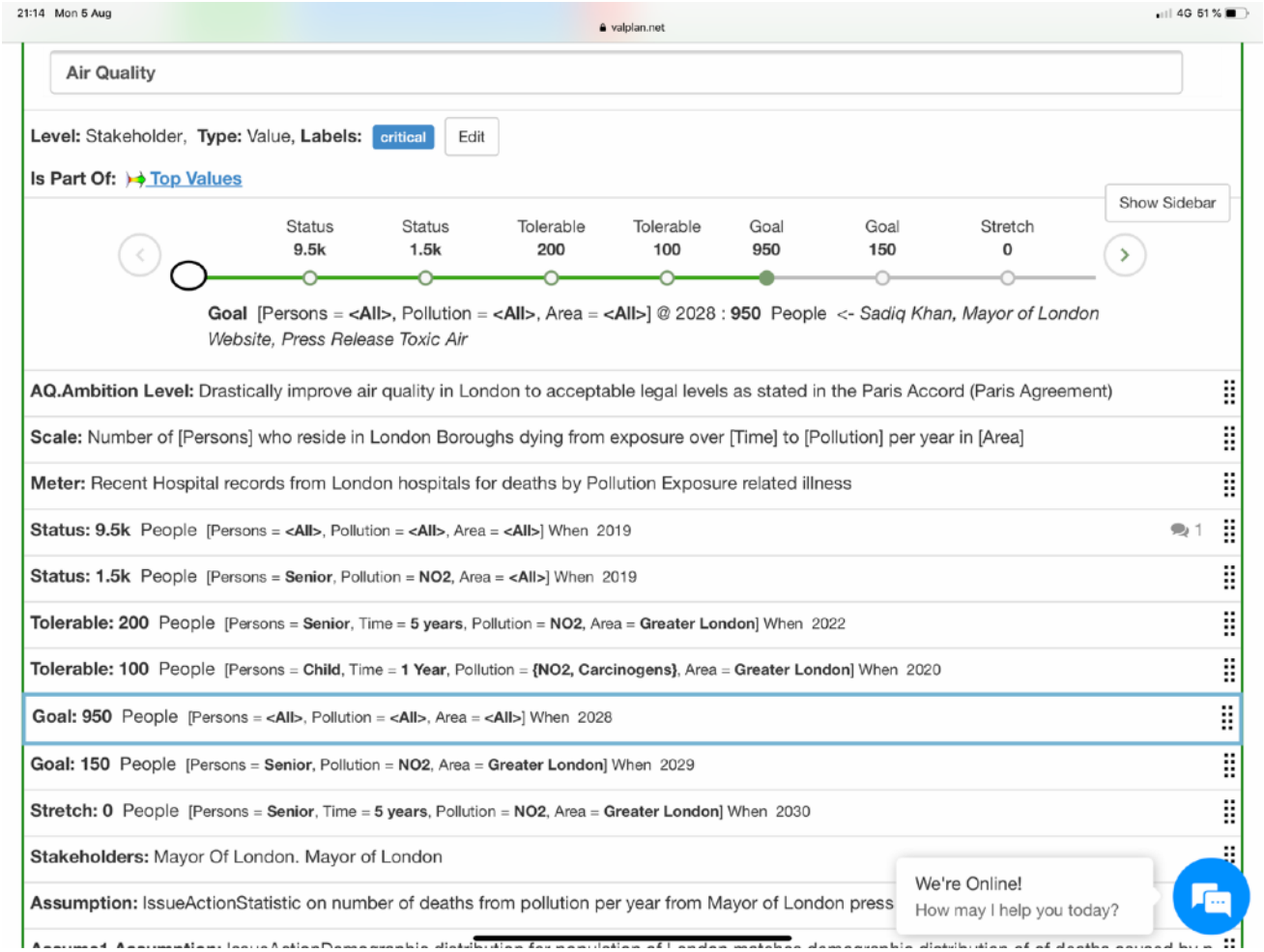


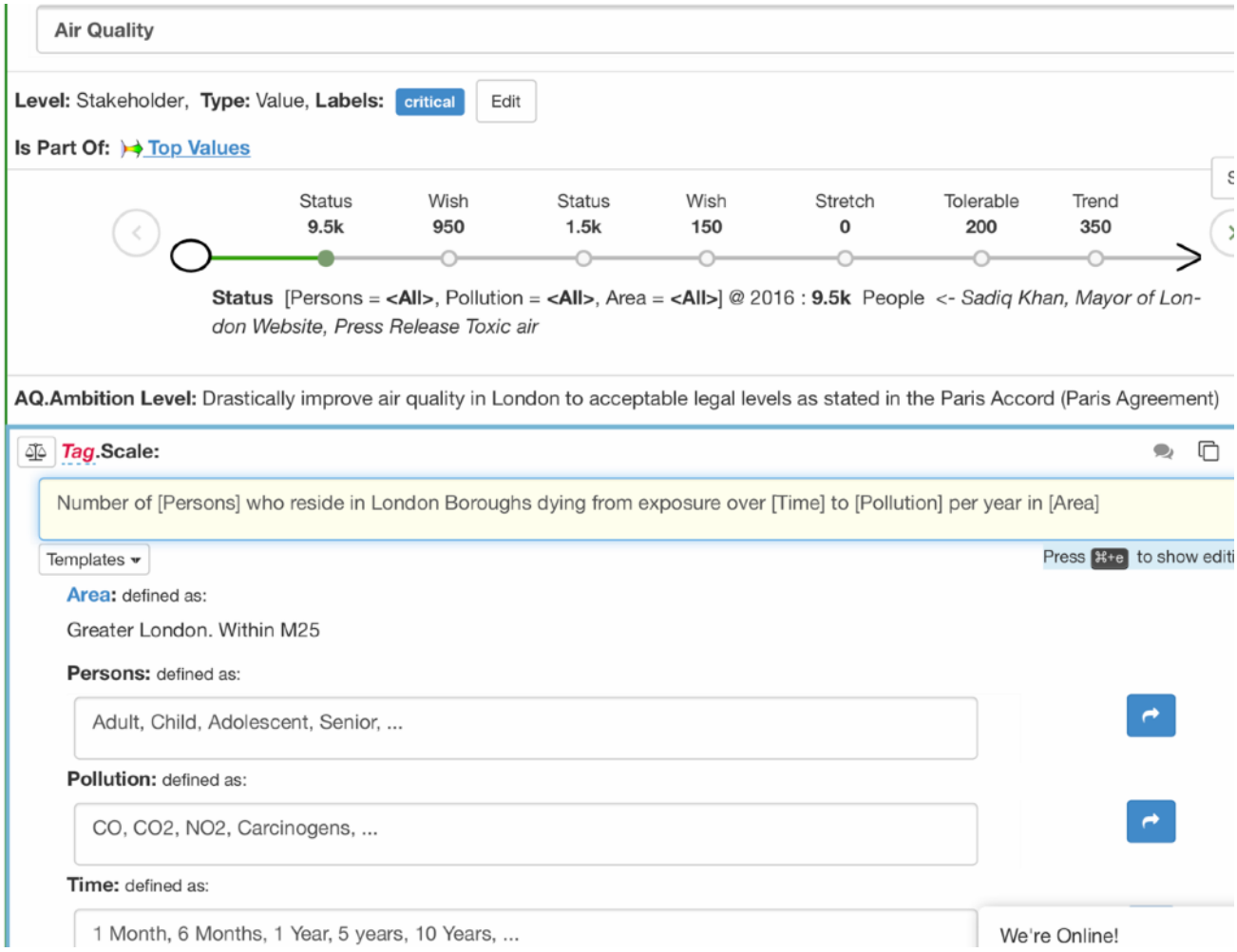
Figure. 1.19 : One of the 11 Values above (Fig, 1.18), Air Quality.

A Summary specification of 1-liners for each parameter. Using Planguage, and the ValPlan.net tool.

There are a few parameters in this example, which are not yet explained in the book. But they will be explained.

But you can guess, look them up in the book glossary, and read them with *some* understanding.

3.10 Details of a Scale (Air Quality example),



with Scale Parameter Definition.

Figure 1.20 : Here is a view of the 'Air Quality' value spec with detail of the Scale. You can see that the Scale parameters are defined as a *set* of attributes.

The 'Area' Scale parameter was previously defined, and reused here. The colored 'Area' is a hot link to the glossary definition of 'Area'.

3.11 More on Multiple Value Requirements

You might be worried about the advice to put aside, for the moment, some of the critical values, in excess of about 10 of them.

They will not be not forgotten! They are usually specified at some level of detail in the overall planning. They are just intentionally delayed in time, so that we have a better chance to actually deliver some higher priority values first.

If we try to do everything at once, then nothing will be delivered early, and we increase the risk of total failure, by running out of resources, political or organizational change, or by failing to learn

hidden lessons from the earlier deliveries.

Figure 1.21: I took this photo June 2019 of a Western Norway River. Value delivery needs to be an early and continuous stream of measurable value improvement. Quantifying our values is a necessary first step.

We are not even going to deliver the 'top ten' values, all at once. We are going to collect enough background information (like stakeholders, risks) on them, to further prioritize some of them.

We are going in the direction of decomposition of both 'values', and decomposition of their technical 'solutions', needed to deliver those values, so that we end up with very early, and frequent, small (2% of project resources at a time) value delivery steps. A 'value stream' to stakeholders.

We are in a hurry to deliver critical real stakeholder value very early, as a continuous stream of stakeholder value results.

We also want learn about the complex environment we are working in, so that we can apply those lessons forward; and to build up our credibility, with the powers that be, for real value-delivery,.

3.12 Knowing when to decompose a value into sub-values, using sub-scales.

Many of your attempts to find quantified value Scales, might be delayed by the fact that your value concept is in reality a set of quite different values scales, which have something in common' Love is a many-splendored thing, as the song goes (ref. b)

There is an engineering heuristic that says 'decompose the value you want to quantify until 'quantification is obvious'. This works well.

Sometimes there just seems to be no quantification available, because you are at 'too high' a level of abstraction.

Earlier we showed that the concept of 'value' needed to be decomposed, into many sub-values. Each with their own quite different scale-of-measure.

This is often true the next level down: some of those 10 values are going to need decomposition, before we can make sense of them quantitatively.

It is very common to need decomposition.

We seem to think in terms of complex value ideas: like 'love' or 'beauty'.

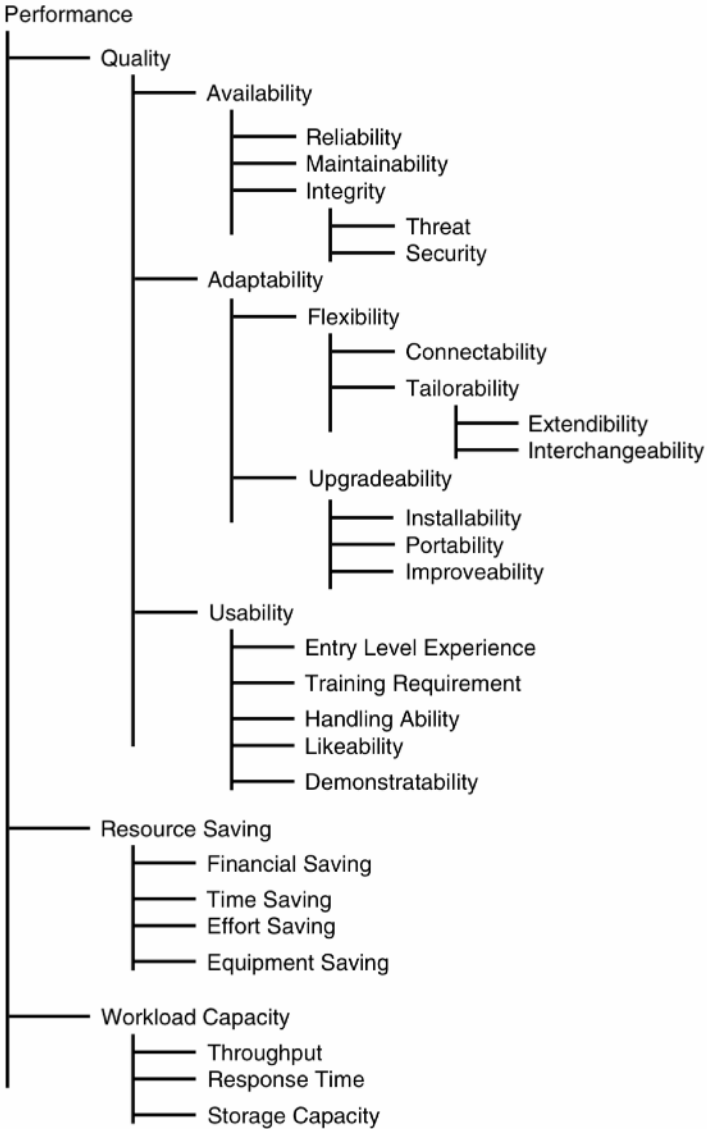


Figure 1.22 : Examples of decomposition of higher level values into different-Scale sub-values.

Detailed examples of potential Scales of measure for these sub-values are given in the CE book. (1)

Some 'values' are simply 'umbrella titles' for a set of different values.

Sometimes, you have an option to decompose into sub-values, or to use Scale parameters to combine the value ideas into one generic scale. Sometimes that is just a matter of taste or convenience. The result might be the same

3.13 Good Scales and bad Scales.

Just because you found a way to quantify a value, with a scale, does not mean you have a good-enough, and *useful* scale.

It means you have 'quantified clarity', and that the *clarity* may even help you understand that is is clearly a 'bad' scale!

Good and 'useful' Scales of measure:

- Are strongly related to the 'values of the stakeholders' who care most about that value.

- Might be more difficult to measure in practice, but you should never choose a Scale *because* it is 'easier to measure'. You must first choose a *relevant* Scale, *then* try to reduce costs of measurement. Cheap measures of the wrong scale are a waste of time.

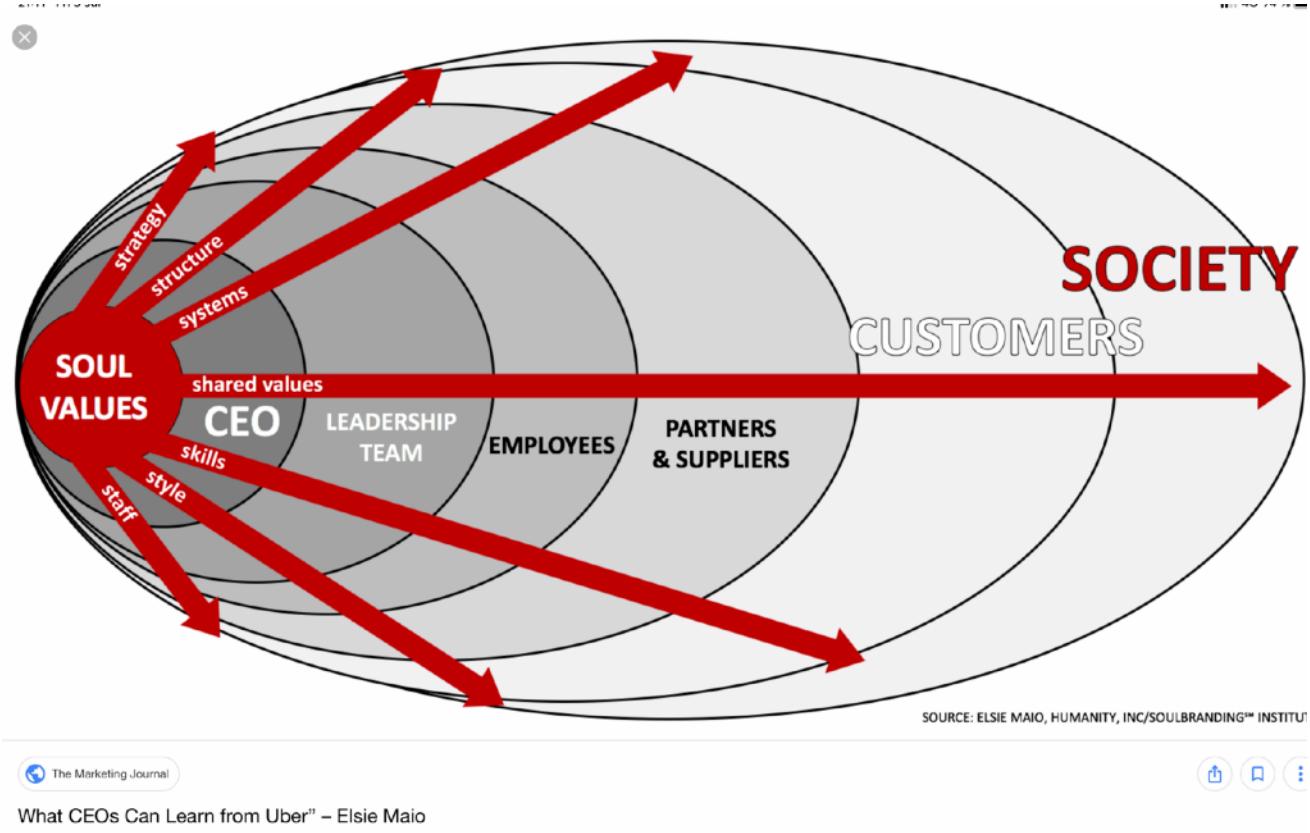
- Will be highly tied to the real environment, with plenty of necessary [Scale parameters] to specify realistic and critical dimensions of the value's application or environment.

Once a client of mine chose 'bugs' counts as their primary quality measure, because they were easy to measure: rather than system (software) availability for the phone system they were building: which they knew was far more critical ('but we do not know how to measure it' they said). This was part of the reason they were 2 years late to market. Micromanaging the wrong value. They got sorted out in time to reduce delay to only 6 months.

By the way, if you have clearly defined a relevant Scale, it will normally not be too difficult to design a reasonable measurement process to suit it. A 'Meter'

The right Scales will feel good and relevant to real stakeholders and domain specialists. Work with stakeholders until they are happy with the Scales.

This is all related to the management interest in 'alignment' of your plans with their values and objectives, at a higher level. Your values must align with the next level above you. Clear real alignment is



the test of relevant value specifications. More later.

Figure 1.23 : alignment levels and related concerns.

Chapter 4. The 'Meter' Parameter

Figure 1.24 : Meters are sort of like this Weather Forecasting Stone. They tell it like it is.

With permission David Bishop (with beard), Photo, Tor Gilb, Hvitsten, Norway, 2019

4.1 The 'Meter' specification is a defined process for measuring the numeric value level, on a Scale.

A Meter Specification is not normally a 'requirement[3]' it is a way of measuring the delivery level of the requirement in a project.

The Meter is very directly connected to the defined 'Scale' of measure. The Meter must measure exactly what the Scale defines. That includes all its [Scale Parameters].

The 'Meter' question is not merely 'was the value level required finally delivered'?

The really useful Meter will give us incremental progress reports on the emerging value levels. It will be designed to give sufficient accuracy at a low-cost, consistent with frequent use.

There is not merely one single test process for a value. There may well be several for different purposes, with different qualities and costs.

---

[3] a customer, in a contract, can require defined test or measurement processes. In that case they are required.

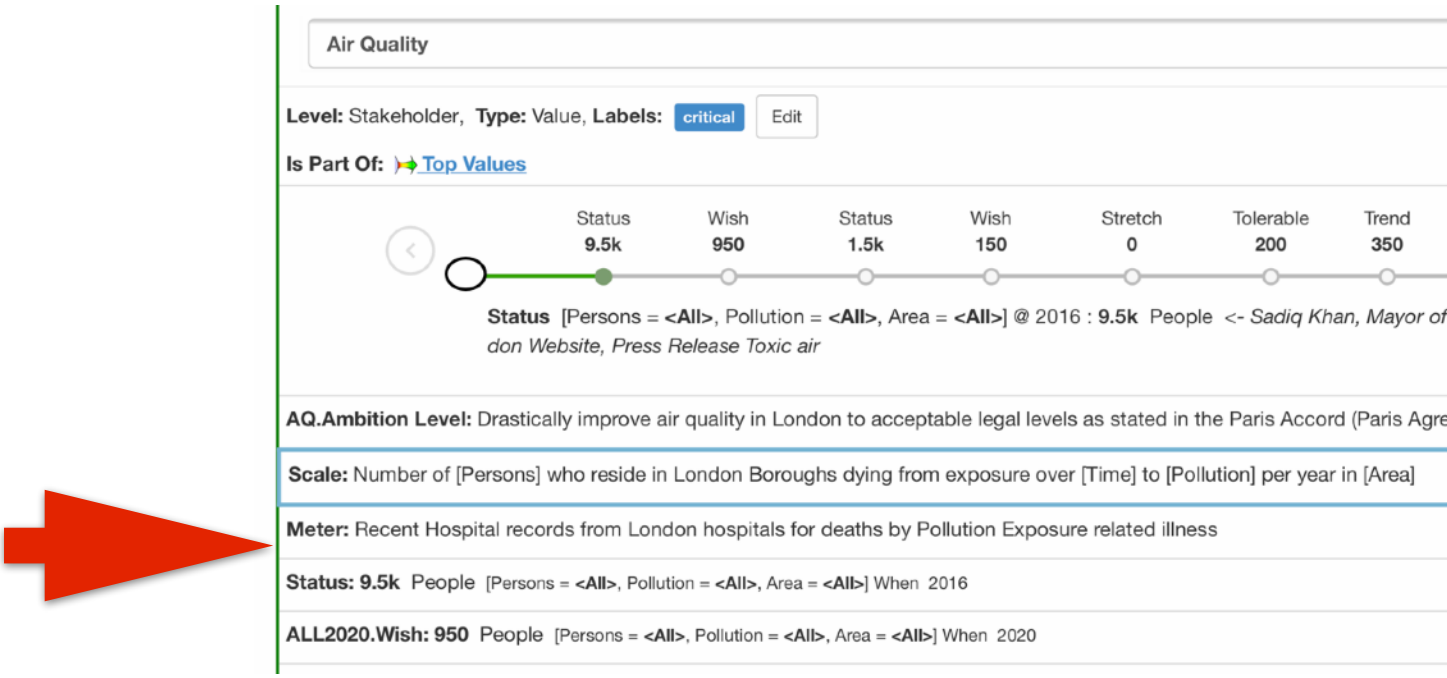


Figure 1.25 : the Meter is a direct reflection of the Scale. It measures along that exact scale.

The Meter has to deliver measurement of all aspects, including all defined [Scale Parameters] of a complex Scale, at a reasonable cost.

A Meter has to have necessary qualities such as acceptable levels of automation, accuracy, credibility, repeatability, setup costs, and legality.

At this 'requirements' level of specification, we might simply *outline* some major ideas of how to measure, and leave the final decision and detail to a professional test planner.

The most critical aspect of a requirement is the Scale and the future required levels.

It is not strictly necessary to define Meters immediately, unless they are contractually required. The measurement process can be

worked out later when we need to measure the value created. But it is useful to sketch a reasonable possible process.

4.2 The Meter as a high-level test process: why this is useful.

I do not practice detailed test planning in the Meter specification. The details should be worked out by professional test planners. In fact they should be able to improve upon, and override a Meter specification.

The purpose of a Meter specification at this early, requirements stage is:

- To suggest that reasonable measurement methods exist at all for this value
- To suggest the possible accuracy, credibility and costs that the measurement process would give us
- To make it clear that we are seriously intending to measure the values delivered
- To give detailed test planners something to start with
- To make it clear if there are any mandatory constraints in the test as part of the system requirements
  - Privacy concerns
  - Contractual requirements regarding measurement for payment

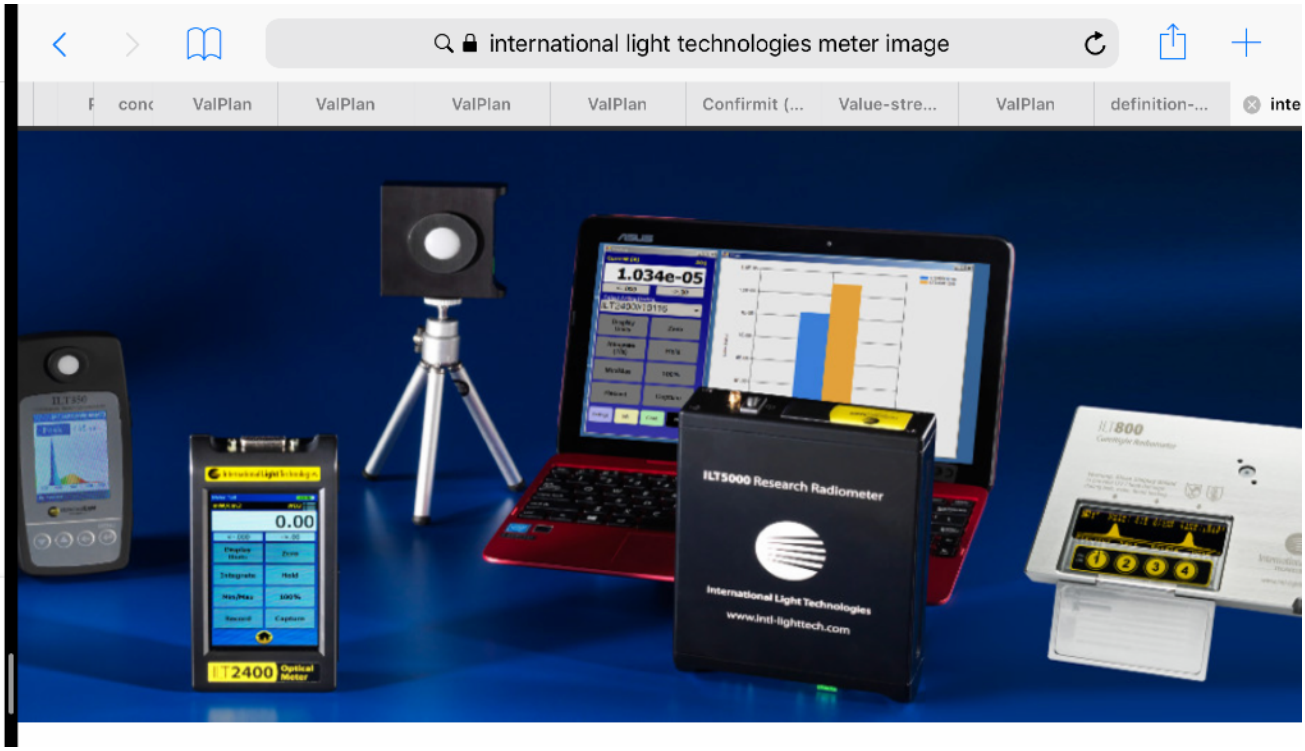


FIGURE 1.26: METERS.

4.3 The multiple quality and cost attributes of a Meter

A Meter, like any test process, has a number of interesting quality and cost dimensions.

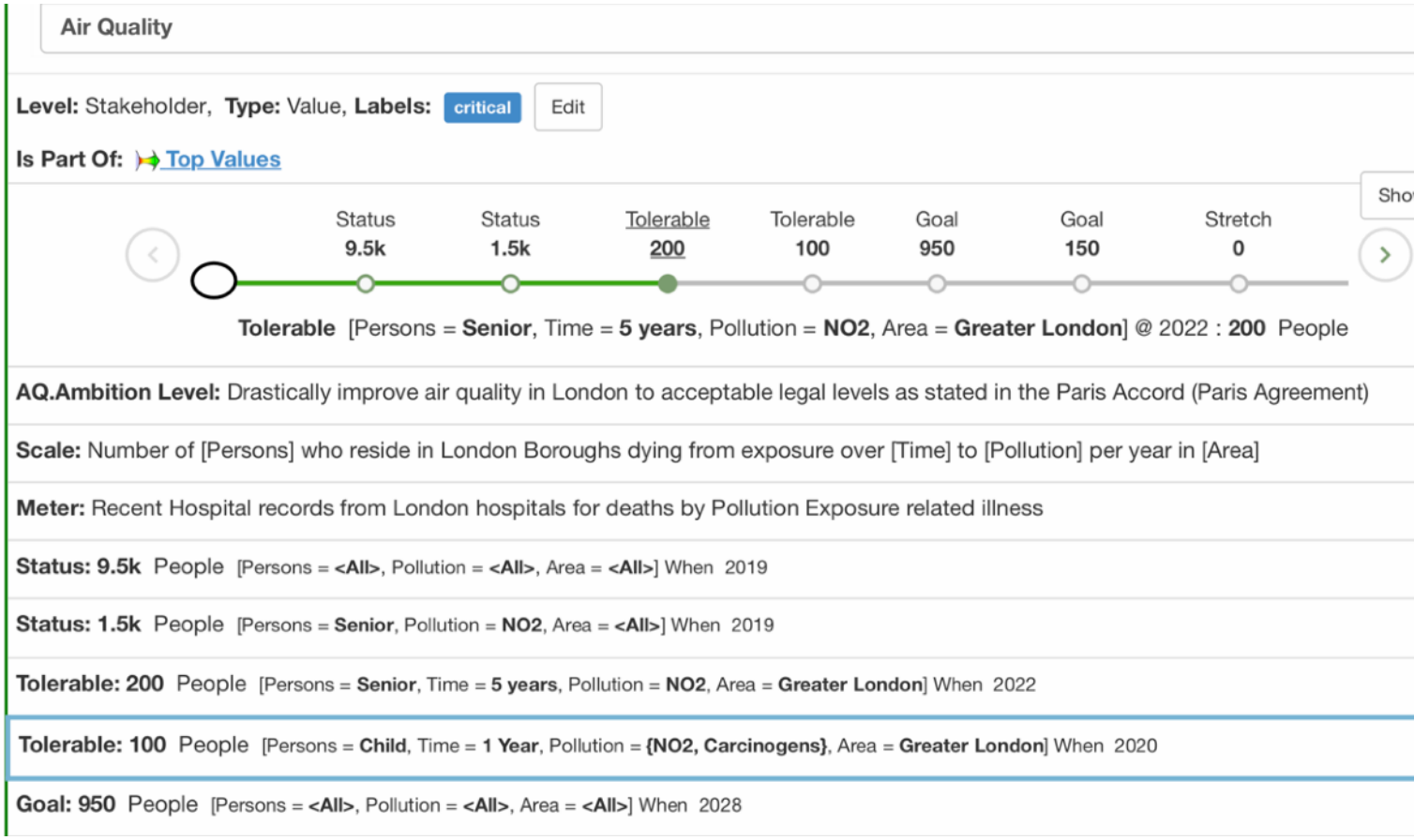
The qualities must be sufficient for purpose, and the costs should be as low as possible, for a defined set of required Meter qualities.

In a sufficiently advanced project culture it might be useful to quantitatively state the Meter Value-Requirements (like 'accuracy'), and to *design* a Meter to be within them.

At least, there is always the possibility of designing the tests to use the least possible resources: for example by using sampling, or automation.

Here are some of the quality Aspects of a Meter:

- Accuracy (is it close enough to the truth?)
- Relevance to its Scale
- Repeatability (same results each time)
- Sensitivity (to disturbing factors)
- Credibility (will people believe it and buy in)
- Legality (will it break laws, customs, standards, contracts?)



- Automate-ability
- Privacy (permission to snoop?).

Here are some of the cost aspects of a Meter:

- Detailed Planning costs
- Execution Costs
- Result analysis costs
- Presentation Costs
- Permissions costs

- Travel costs

4.4 Sufficient Meter Accuracy for Purpose

In early incremental stages of value delivery, high quality measurement is not generally required.

It is sufficient to be pretty sure things are moving towards the required levels of the value, at a reasonable pace. At the extreme, 1-digit accuracy of the % value-improvement might suffice.

One client of mine dropped measurement of weekly increments, and left it to the very-experienced intuition of the system developers. At an earlier stage the same client decided to use no more than 30 minutes per weekly increment, to measure value delivery. For Usability factors they even got lucky when Microsoft Usability Labs offered to measure weekly, overnight, for free. (Ref. C,D). They did take release quarterly of product upgrades far more seriously for measurements.

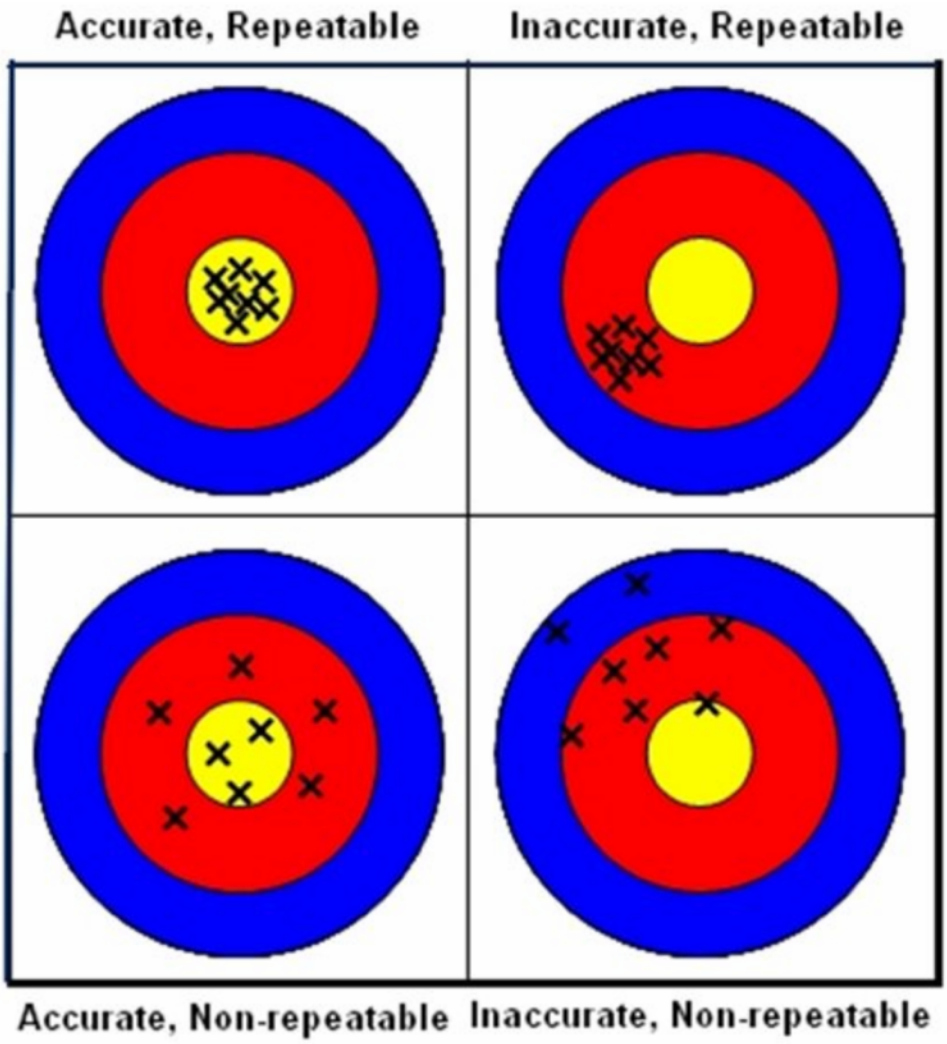


Figure 1.27 : Accuracy and other Meter concepts.

Chapter 5. Benchmarks

5.1 The purpose of 'Benchmarks' In a Value Requirement Specification.

A 'Benchmark' level of value, on a Scale of measure, is *background* information about a requirement level. It helps us decide if we have set the real requirement levels appropriately.

This is traditionally something a Business Analyst should look at, as a prelude to setting requirements.

But in Planguage, I decided that it was better to *integrate* Benchmark data with the requirements data.

- in order to make it possible for all reviewers and creators of a requirement object, to decide for themselves if the requirement levels are in reasonable proportion to the benchmarks

- To make it even clearer if the Benchmarks data is missing, or not particularly credible, or up to date.

- To support incremental delivery, where Benchmarks need to be updated, at each increment, not just in an initial Waterfall analysis phase.

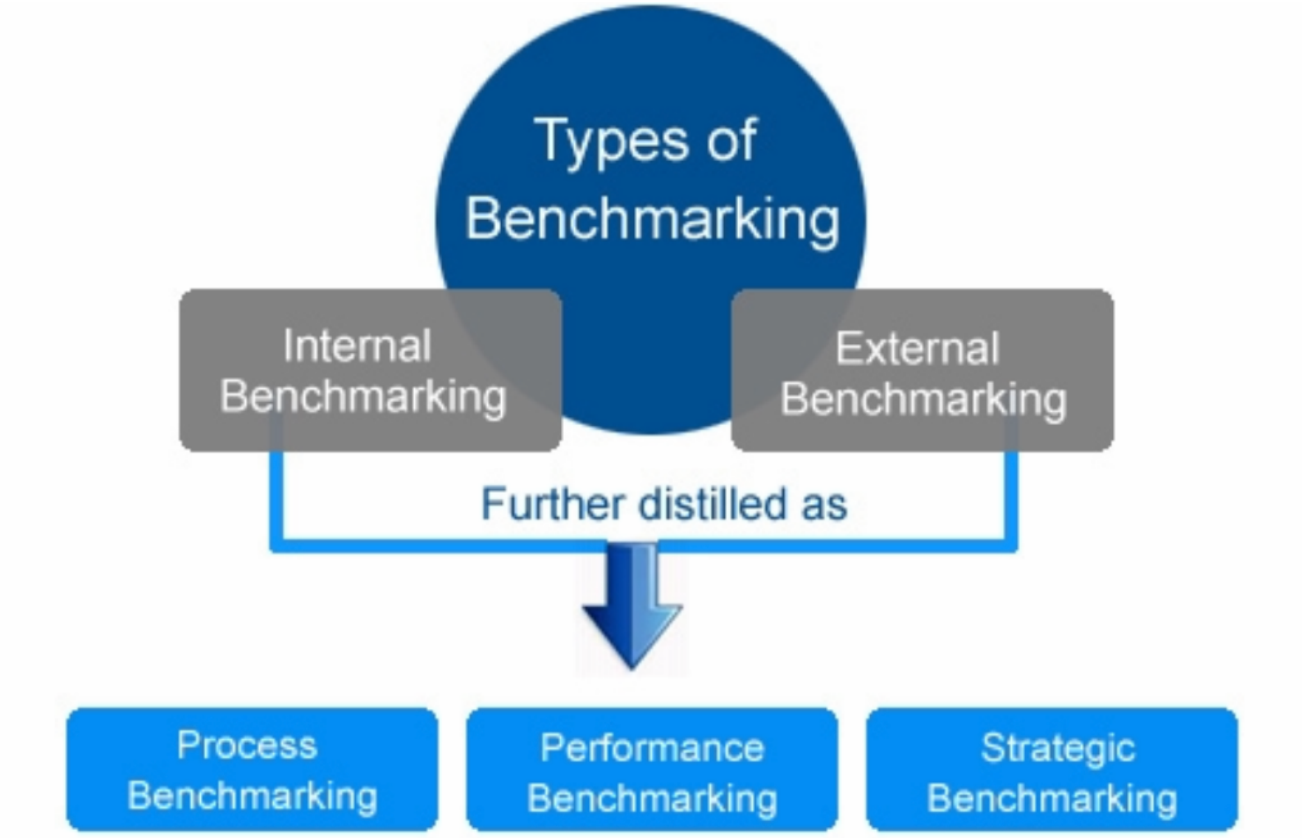


Figure 1.28 Types of Benchmarking.

Figure 1.29 : 2 different 'Status', which is a type of benchmarking. Benchmarks are 'background' information embedded in a requirement.

5.2 'Past' as a Benchmark

A Past level statement is a fixed result at a fixed date. History of a level which happened.

You can insert as many Past statements as are potentially useful, at any time in the process. As new data occur for example.

Using Past level information we can better decide if our requirement levels are appropriate.

- are we planning to be good enough in relation to our own Past levels ?

- And those of competitors ?

- Is updated Past level information sufficient to force us to reconsider planned requirement level specifications ?

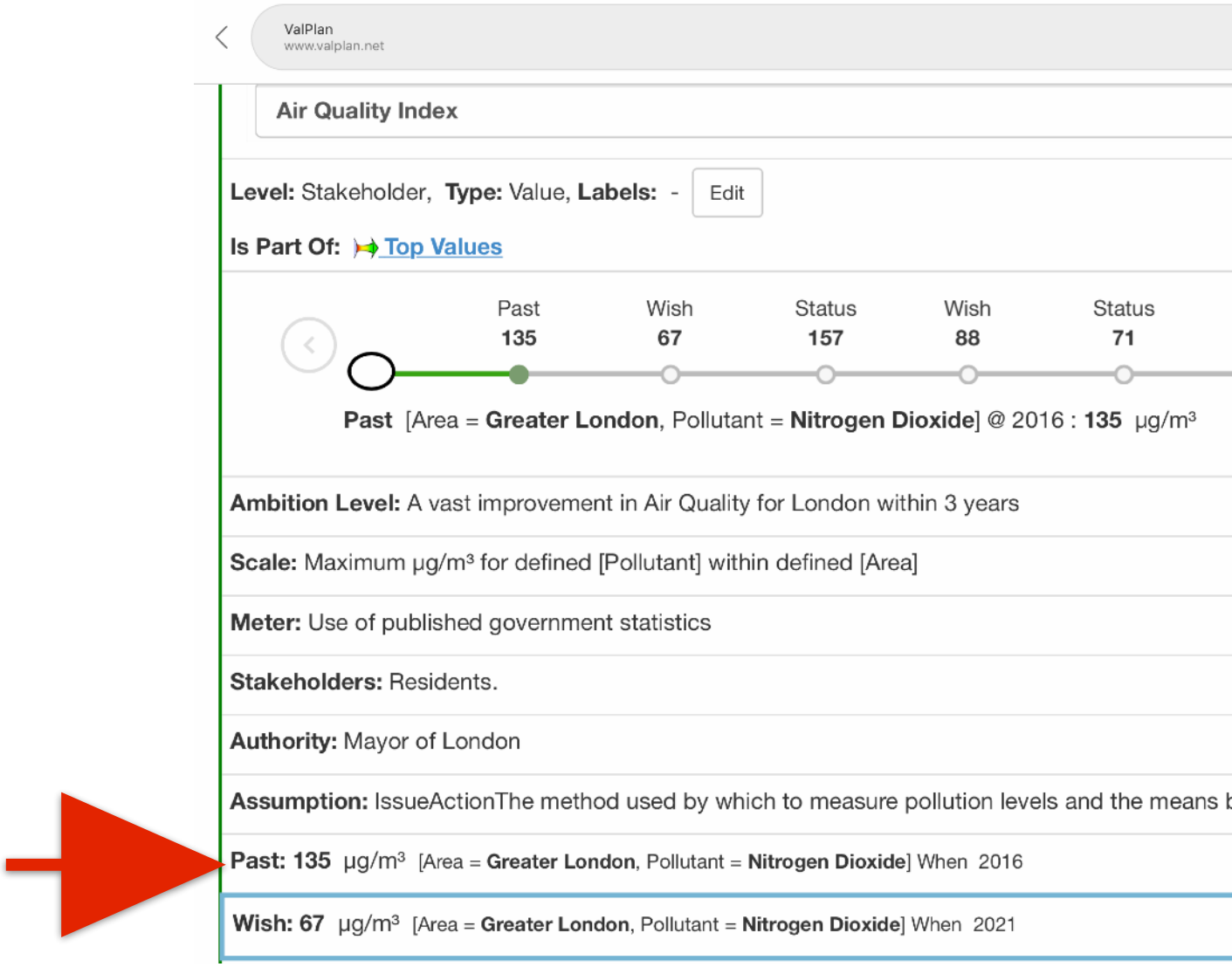


Figure 1.30 : The Past data here for Greater London and Nitrogen Dioxide, are directly comparable in the 2 Scale parameters, and the units of measure on the Scale. The requirement is for a 2x reduction over a 5 year period. As it is now 2019, we need to ask if the Past data is up to date (at least 2018) and if any progress has been made as a result of our project deliveries, if any. It is time to introduce a Status specification to track progress.

Note: I am using examples using the ValPlan.net tool. But this tool is NOT a prerequisite for using this method or Planguage. A Word Processor works fine (1, CE). Just more work.

'Past'-level data is not necessarily from our own systems. It can be from any system that might be useful to compare us with.

Competitors, and other industries using similar methods or architectures.

5.3 'Status'-level Benchmark: real time value delivery tracking.

The 'Status' benchmark level is intended for use in incremental value delivery, to track our own project progress, or lack of it, towards required levels.

It can be used initially as a departure point, for tracking progress on your very own system: an incremental baseline in the continuous learning and re-planning process.

We can keep track of a series of Status, in the basic value requirement. But we can also track status as a graph line, based on feedback in increments after a value delivery for our system. Or both.

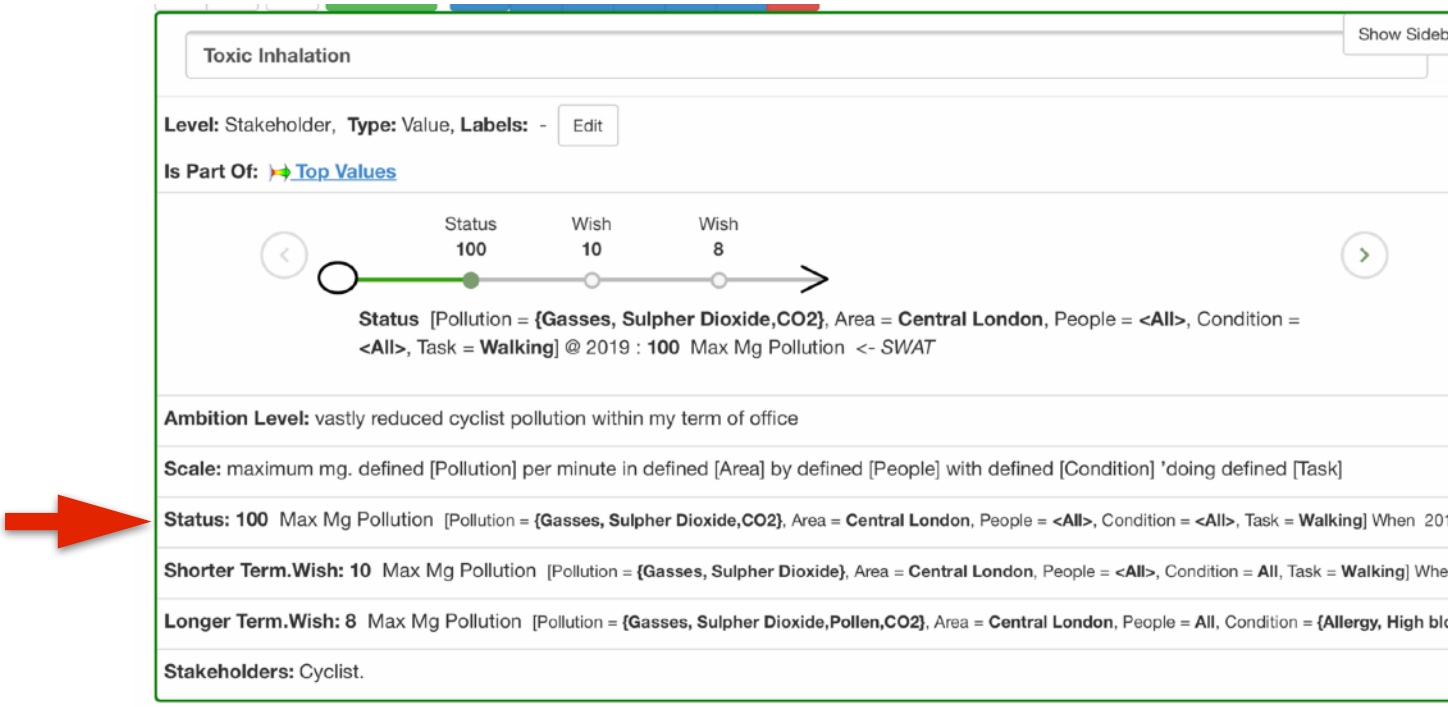


Figure 1.31 : here is Status used as an initial planning data, 'where our system is before we start delivering value increments'. It is followed by 2 *different* Wish levels, which have slightly different Scale parameter attributes, and different delivery dates. So the Wish levels are not completely comparable to the Status information. A signal that Status information might possibly be updated, to be comparable, for those Wish conditions, if possible.

5.4 Record Benchmark

A ‘Record’ benchmark is information about some extreme level of a value, good or bad.

It can be a Record level for us, or for others, like competitors.

The purpose is to stimulate us to be competitive with the best, both of our competitors, and with those in other domains using similar technology.

It is the sign of an expert that they know the Record Levels in their domain.

Keep in mind that Record setters do not stand still, but are probably trying to improve on their record. It is not sufficient to beat the old Record, you win by beating the new Record in the future.

We sometimes try to guess that using the ‘Trend’ parameter spec. (See Ch. 5.6 below)

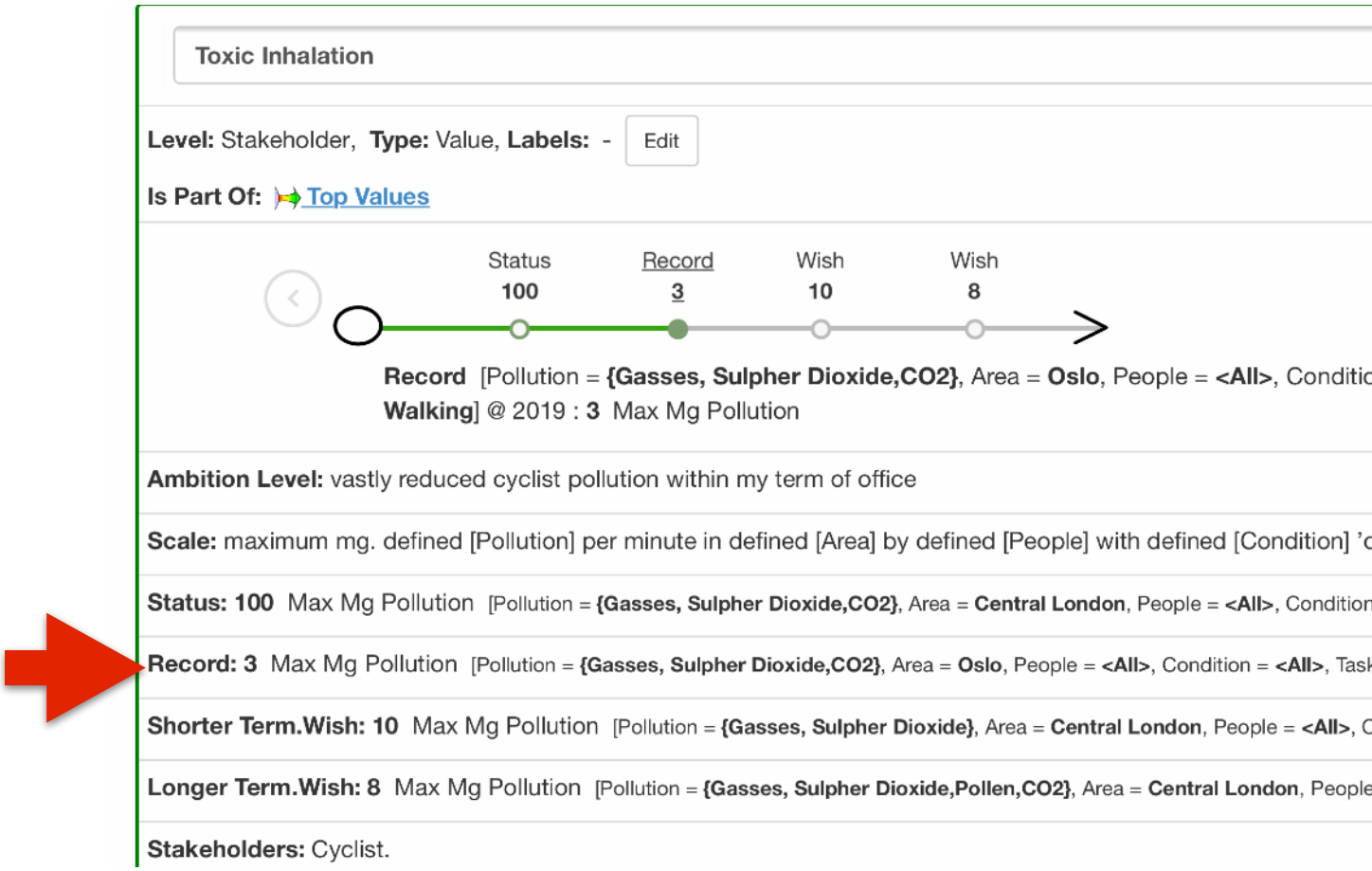


Figure 1.32 : We added a Record, comparable, but for 'Area = Oslo', and it shows that it is possible for a waterside city to get to a value level of 3. Well Oslo is not London, but what are they doing that we might learn from, and are our desired value levels ambitious enough?

The 'Record' levels can be particularly useful, because if you analyze the technology used to reach the Record level, and the costs incurred, you might come away with very useful insights.

5.5 'Ideal-level' Benchmark

The 'Ideal' benchmark is rarely specified, since it is rarely attainable. Perfection tends toward infinite costs. If ever attainable.

So I use this mainly in oral discussion to point out unrealistic ambitions. Unrealistic requirements.

Dangerous if they end up in a contract, as one of my Oslo Tech business clients CEO found out to their horror. They had contracted for 99.9999999% uptime for an airplane phone system with a big international supplier. The CTO when I asked, said no-one in the mother corporation had even done that, or knew how. So we had to 'adjust' the contract, or go bankrupt. They succeeded with the 100 person, 1 year, $20 million project after that adjustment. Actually it was the first time they made a profit in several years. The marketing chief had had no problems saying yes to the customer's 'Ideal' . Salespeople get tempted to promise 'Ideals' which are unattainable. I assume they negotiated a more realistic availability level (like 99.98%).

Case 1N.

People are regularly specifying things like '24/7'. Which sounds like 100% availability to me.

Engineers know they can't do 100%. 99.998% is fine!

If necessary, specify Ideals *formally,* to erase all doubt.

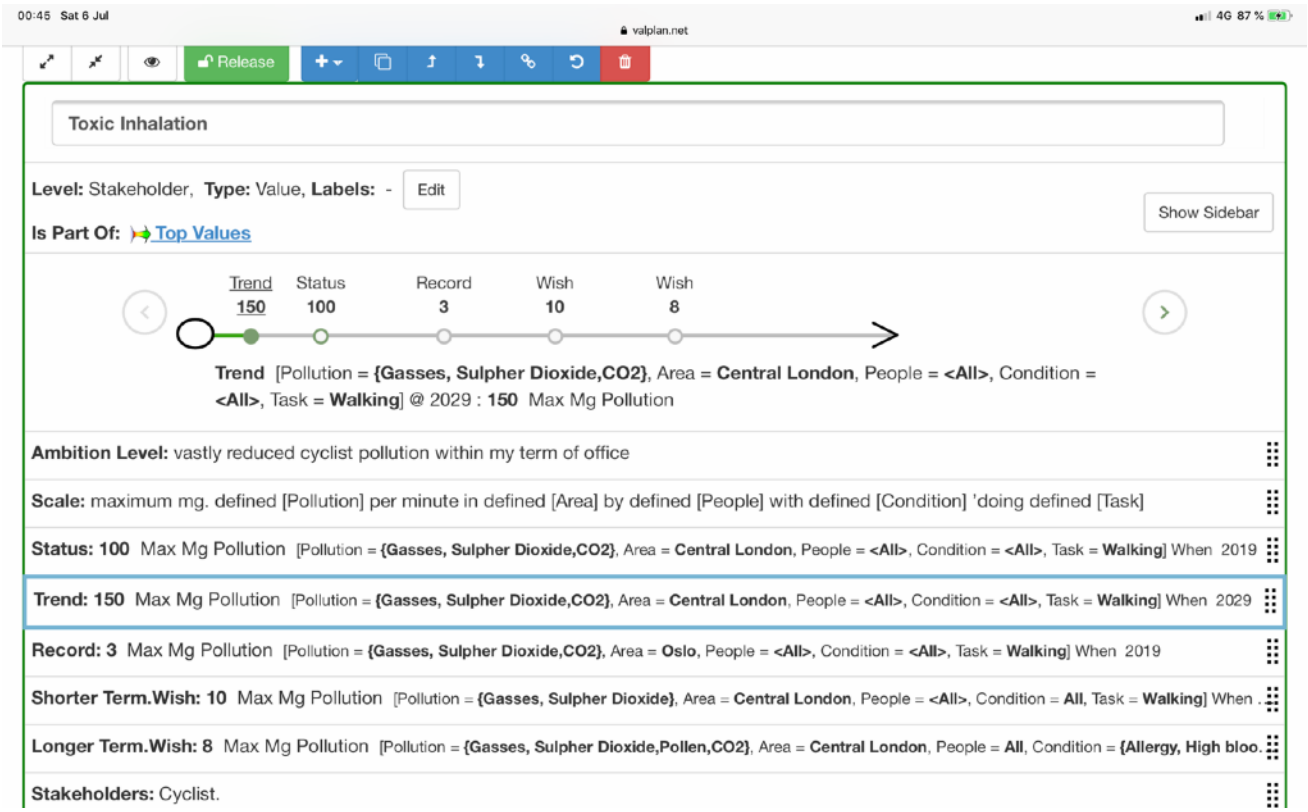
Ideal: ZERO Pollution of any kind in London, ever and forever. Not planned yet!

Example 1O.

5.6 'Trend-level' Benchmark

The 'Trend-level' benchmark is an attempt to stop looking in the rear-view mirror, and look out at the road traffic, coming up, *ahead* of us.[4]

This is especially important in environments which experience high rates of unpredictable change, from competitors, enemies, nature, economics, technology, and politics. That is just about all of us



today, I guess.

Figure 1.33 : The 10 year Trend, if we do not act, is 50% worse pollution. Useful to remind people of the alternative to funding and supporting your project. Don't assume people know such things. Research it and spell it out explicitly.

---

[4] Kai Gilb invented 'Trend', in connection with Ericsson assignments. Looking ahead is very important is fast-moving competition.

6. Scalar Constraint levels.

6.1 'Tolerable-level' Constraint.

Formal Planguage definition:

http://concepts.gilb.com/definition-Tolerable-Limit

The 'most critical' value requirement level is the 'Tolerable' level.

It defines the borderline between **failure** (below Tolerable,

Intolerable) and **not-failure** (Tolerable).

Figure 1.34 : the Tolerable Level, or Tolerable Range is just above the 'Intolerable level, and is a range extending until a 'success level' is defined. It is possible to have Intolerable levels and ranges at both extremes of a value scale, as in too hot and too cold.

Setting such constraints is mainly subjective. There is only rarely a 'cliff edge' at that point. But it is better to have clearly-defined fail/not-fail borders, than to leave your team in confusion about the borders.

This can easily have contractual implications, and you don't want to pay legal staff to argue in court about the meaning of 'sufficient' just because you did not make up your mind in the first place, in the requirement.

People would be wrongly motivated if they focused on just getting barely to the Tolerable level, at the edge of the border. Their main motivation *should* be:

- To get *well clear* of the Intolerable area, quickly, immediately.
- To create a safety margin by being *well-above* the borderline.
- To relax further efforts here, this particular value, until all other critical values, were also well clear of Intolerable dangers.
- Then to march on, towards target levels, like Goal, which define *success*.

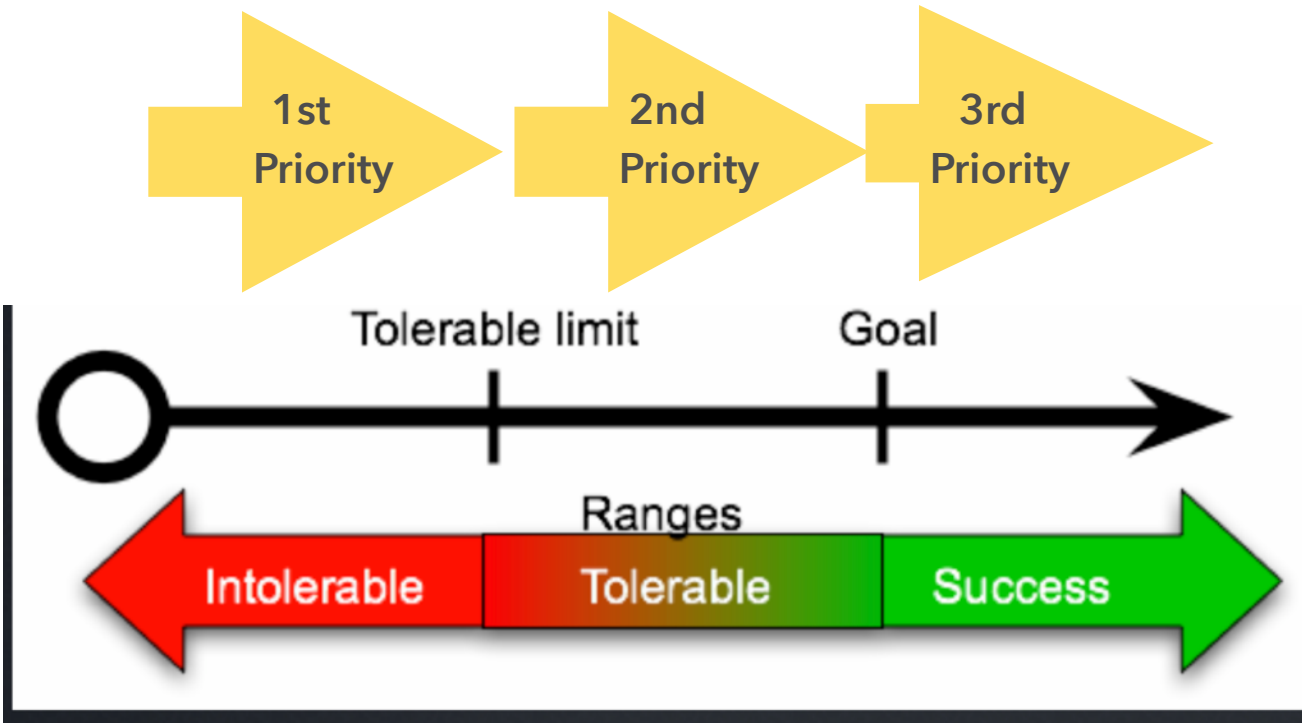


Figure 1.35 : Getting out of Intolerable levels of a critical value, is our first step. Then incrementing the value until we reach all success target levels (like a Goal). If any resources remain, we can choose to use them to increment values to more than 'just barely' success. Perhaps towards Stretch levels, or to longer term, and special conditions, success levels.

Because our top-level critical values are 'critical', meaning 'critical for the entire system, product or service', then as a rule, if even only one single top-level value requirement, fails to reach the Tolerable level, this probably implies failure of the *entire system*.

As a simple example, if all other top-level critical values are Tolerable or better, and the availability of the system is below the Tolerable level, say nearer 0%, then by definition none of the system functionality is available, most of the time. And none of the other value attributes, such as work capacity, usability, and security, are available. So this describes total system failure.

The determination of the Tolerable Level is a matter for the relevant stakeholders, and their practical needs and experience. At

what point do they throw up their hands and say "I give up", and use alternative ways to satisfy their needs?

The Tolerable Level might also be set by other stakeholder needs such as legality, conformance to standards, economic profitability, or a first rough guess at the right level.

6.3 Constraints are a Dynamic Prioritization Tool

Another insight into applications of the Tolerable Level is that it is a powerful tool in helping us manage priority *dynamically*, that is, managing 'step by step as we deliver value'.

Once we have reached a single Tolerable level, we need to ask ourselves (project management) if we should ease off on delivering more value, just yet, to this particular value.

We need instead to ask if any *other* critical values are still under their *own* Tolerable levels, and divert resources immediately to the task of getting *all critical value requirements* to at least Tolerable levels.

We need to get the system into 'Tolerable conditions' with respect to all critical values, *before* we plunge forward to satisfying Target levels for the critical values.

Figure 1.36 : Sailing requires dynamic prioritization. Source: "To Catch a Butterfly: Epistimic Miracles of Serendipity. The.xel.io

http://te.xel.io/posts/2018-03-04-to-catch-a-butterfly-epistemic-miracles-of-serendipity.html

6.4. Several different Tolerable levels might be appropriate for different circumstances.

One single value might well need to specify a variety of different Tolerable levels for different circumstances.

This avoids over-generalization of requirements, with consequent unnecessary costs for some circumstances.

And it supports our need to focus on particularly critical circumstances early, delivering value to those circumstances early.

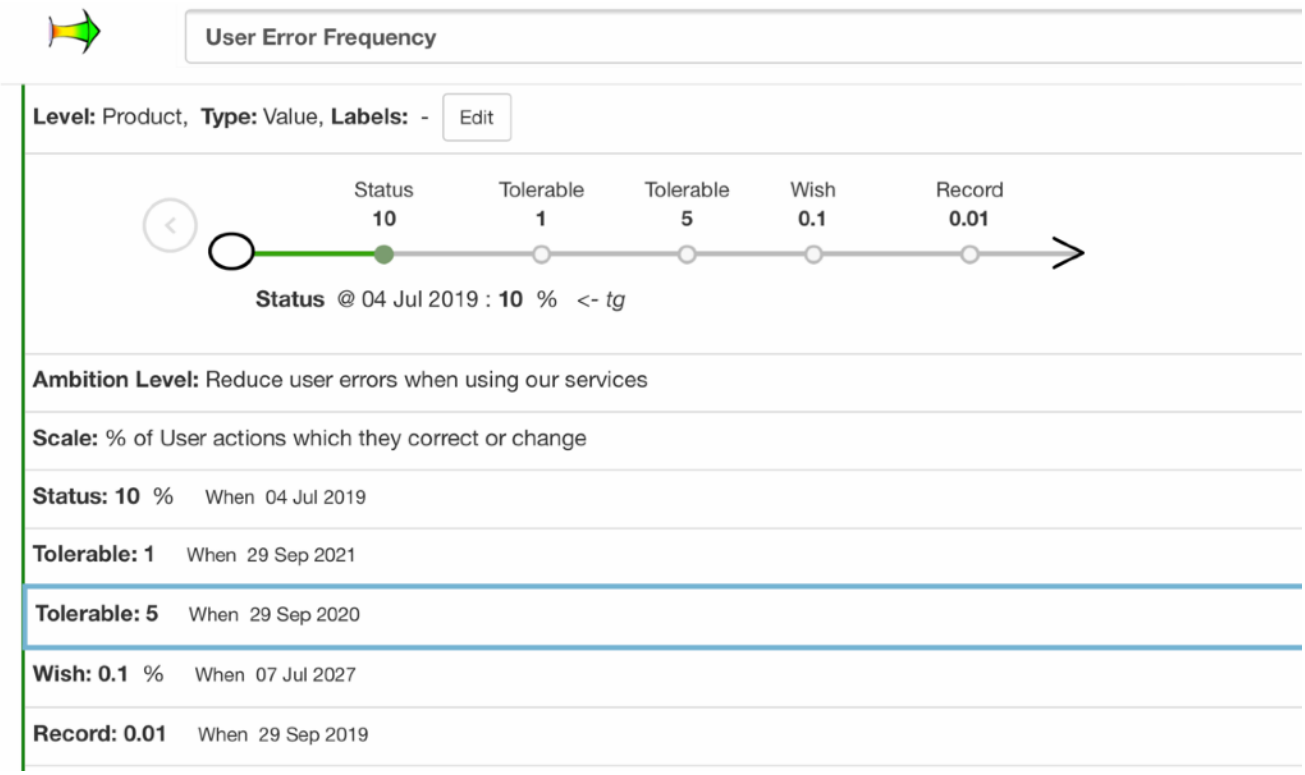
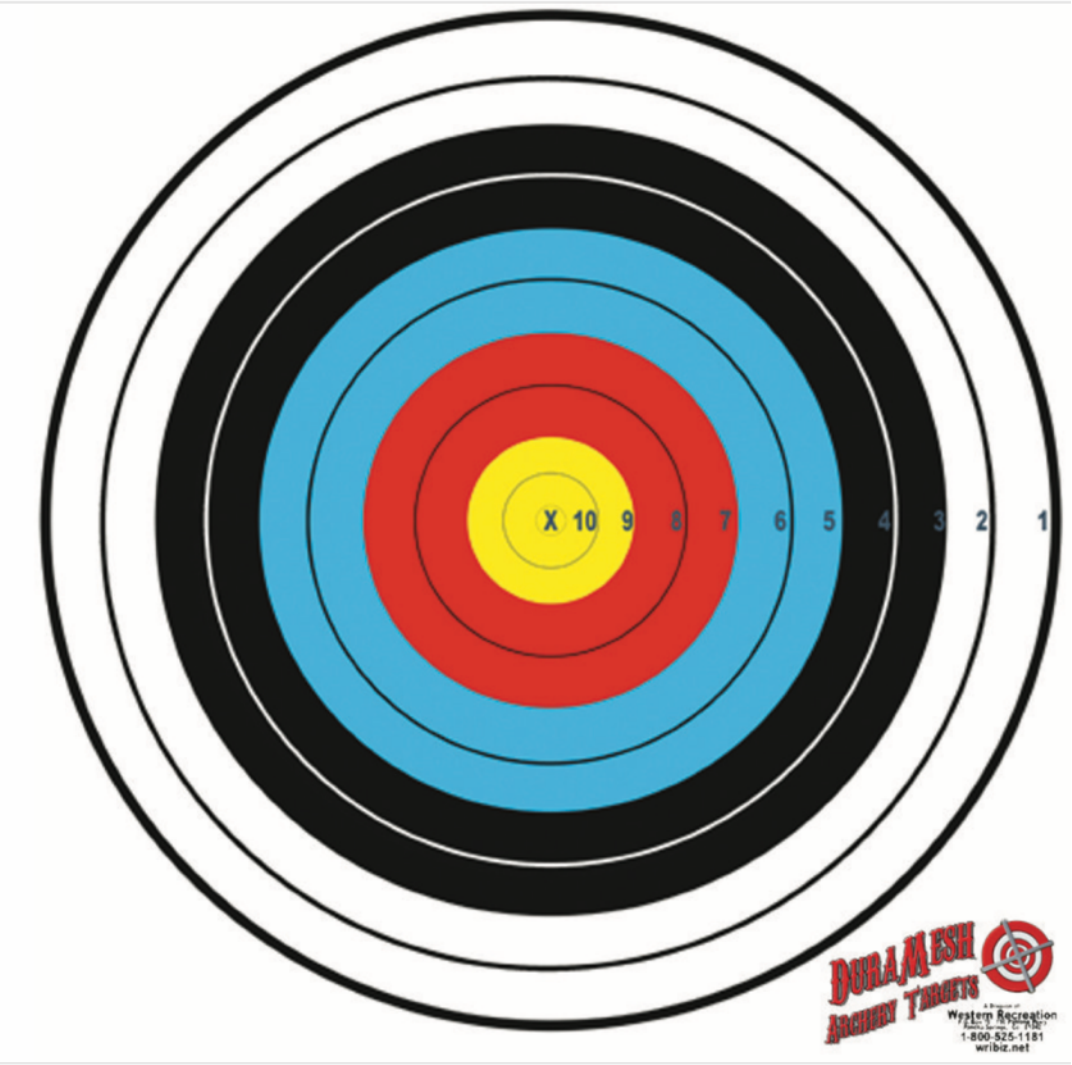


Figure 1.37: example with 2 Tolerable levels, with different deadlines.

6.5 There are *other* types of 'Scalar Constraints' Defined

In my books, (VP, CE) but this one is sufficient for most purposes.



Chapter 7. Scalar Target-levels.

Figure 1.38 : a target level is a value level we positively aim to achieve. There are varying degrees of hitting the target. And there is such a thing as *not hitting the target at all*.

7.1 Wish-level Target

A 'Wish' specification is an expression of a *stakeholder desire*, based on *their* needs and values. It is a '*stakeholder* target', but not yet qualified as a '*project* target'.

'Wish' belongs to systems analysis: what should this project *consider* delivering? What would be *valued most* by the important stakeholders?

There can be serious problems with Wish statements, which means we cannot simply accept them as serious *project* requirements.

'The customer is always right', but they might not know state-of-the-art limitations or have infinite time and money.

Stakeholders are allowed to dream and be ambitious, but not every Wish is realistic, or is not consistent with other stakeholder needs of higher priority.

- They might be *unrealistic*, technically and economically.

- They might *steal resources* from other more-worthy requirements and stakeholders.

- They are usually expressed *without* the stakeholder having *any overview* of all *other* Wishes and constraints.

Wish statements are our formal acknowledgement that we have analyzed the stakeholder needs, and recorded their desires.

But they cannot simply be considered serious project *requirements*.

They need to be analyzed, for technical feasibility and economics.

Then they need to be prioritized together with all other Wishes (and Goal commitments), as part of the overall system, overall economics, and overall priorities.

When 'Wishes' pass all necessary tests, feasibility, economics, priorities - they can be converted to seriously committed requirements. Like 'Goal' specs.

To commit immediately to 'User Stories', and Customer Requirements, just because we want to respect them, is not wise.

It can lead to broken promises and hurt feelings, and even at the extreme, total project failure. That is not true respect. We have to be realistic, and we have to prioritize: *always*.

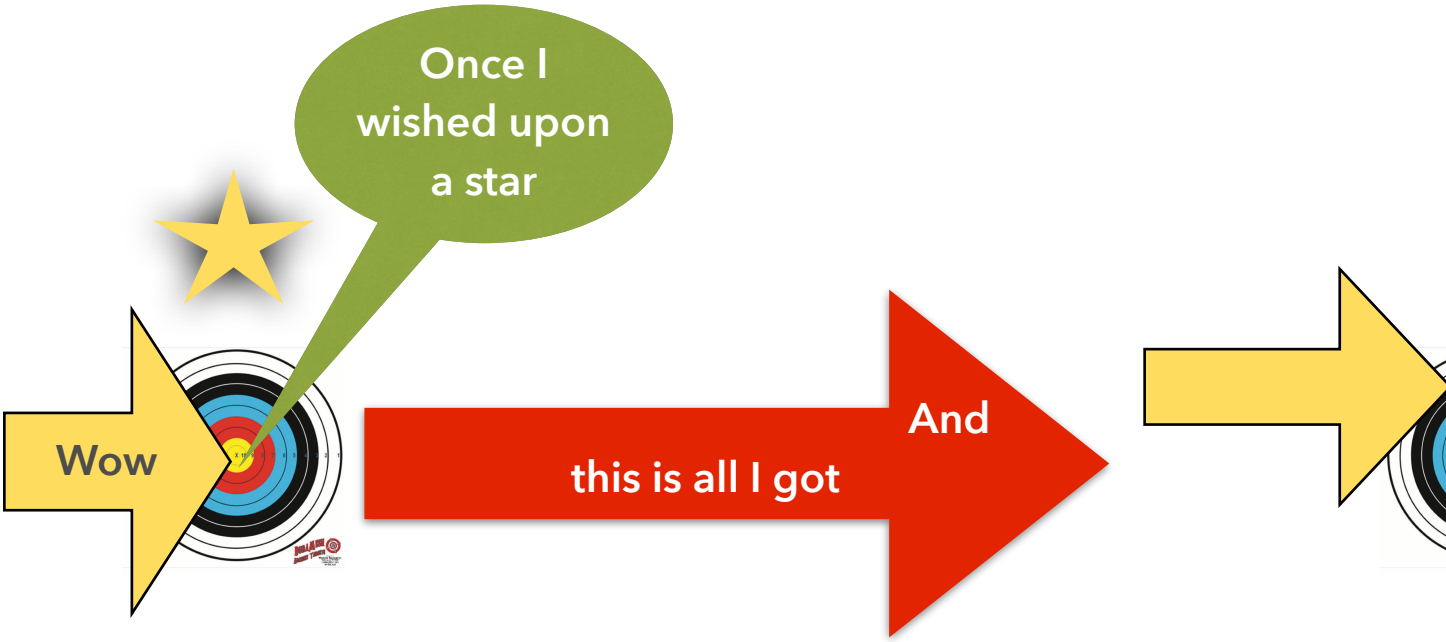


Figure 1.39 : "Santa, I want a real Tesla X for Christmas, not a toy."

The Wish must be a clear and detailed enough.

It is not good enough that the wish is almost always 'Wishy washy', highly ambiguous, like a typical Ambition-level statement, or a User Story. (see Ch. 15.1)

This is because you then, cannot really understand what is being asked for, and therefore whether it is *possible*, *economic*, and what its priority is.

So the Ambition Level still has to be translated into a numeric and well-structured Scale, as discussed above.

And that is not all. You have to decide exactly which Scale parameter attributes (who, what, where, and when) need exactly

'how much value level'. If not you still have a fuzzy question, and you are not going to get unrealistic answers.

For example: if the Stakeholder says:

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

Example 1P

Or

User Story: As a User I want good security, to fight bad guys.

Example 1Q

These are simply unacceptable statements:

Their possible range of value, and consequent technology interpretations, is far too wide. The cost range is roughly zero to infinity.

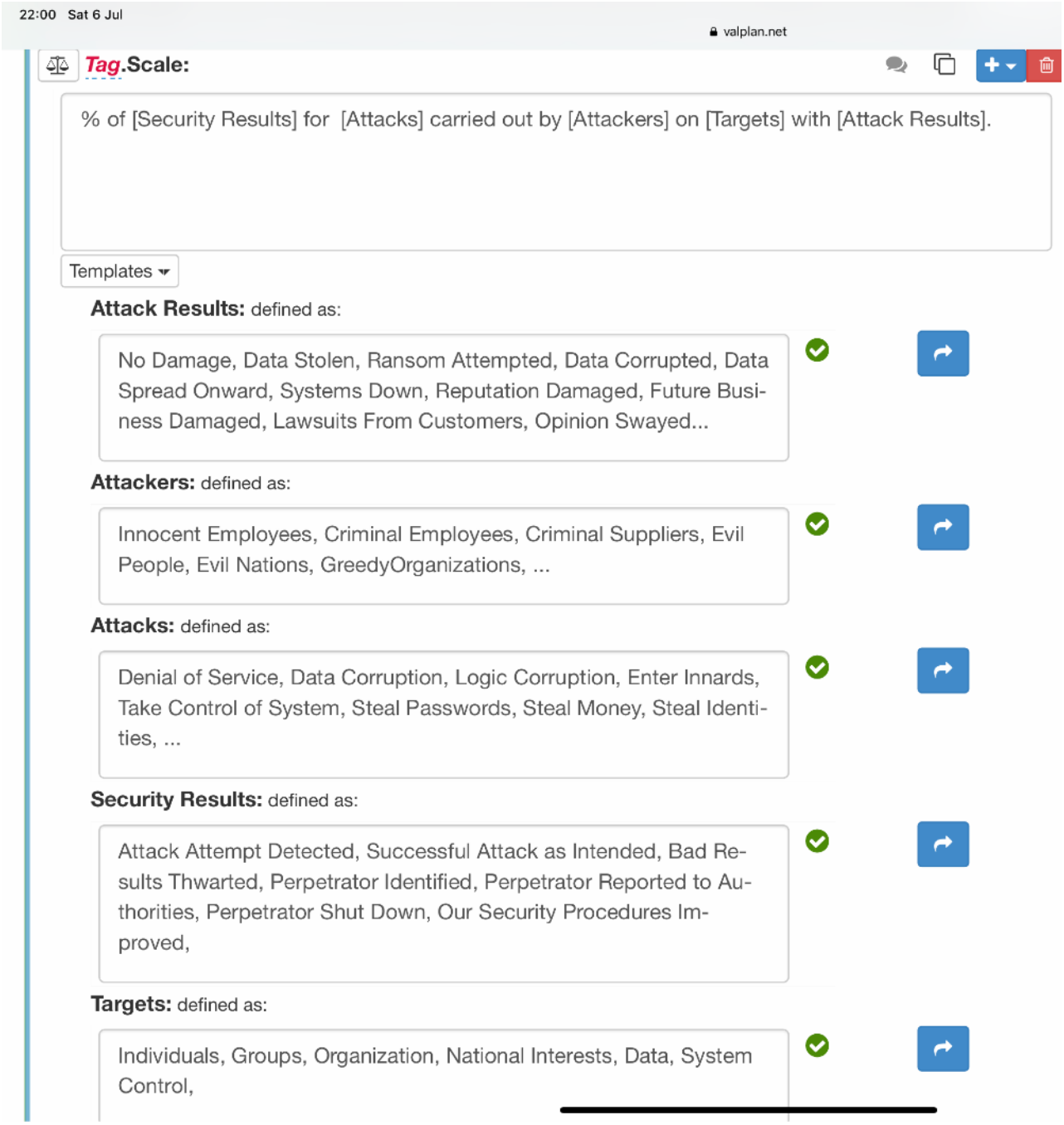


Figure 1.40 : in this case I took the Ambition Level statement ("I want the best security to fight hackers and protect my customers and company") , and created a Scale for Security with appropriate Scale Parameters ('Attack Results', etc.).

I then defined all 5 Scale parameters (Fig. 1.40) with a reasonable set of attributes. Anything forgotten can easily be added later, as we go.

Can you begin to see the need for detail in this Security problem?

The ‘Ambition Level’ hides all of it.

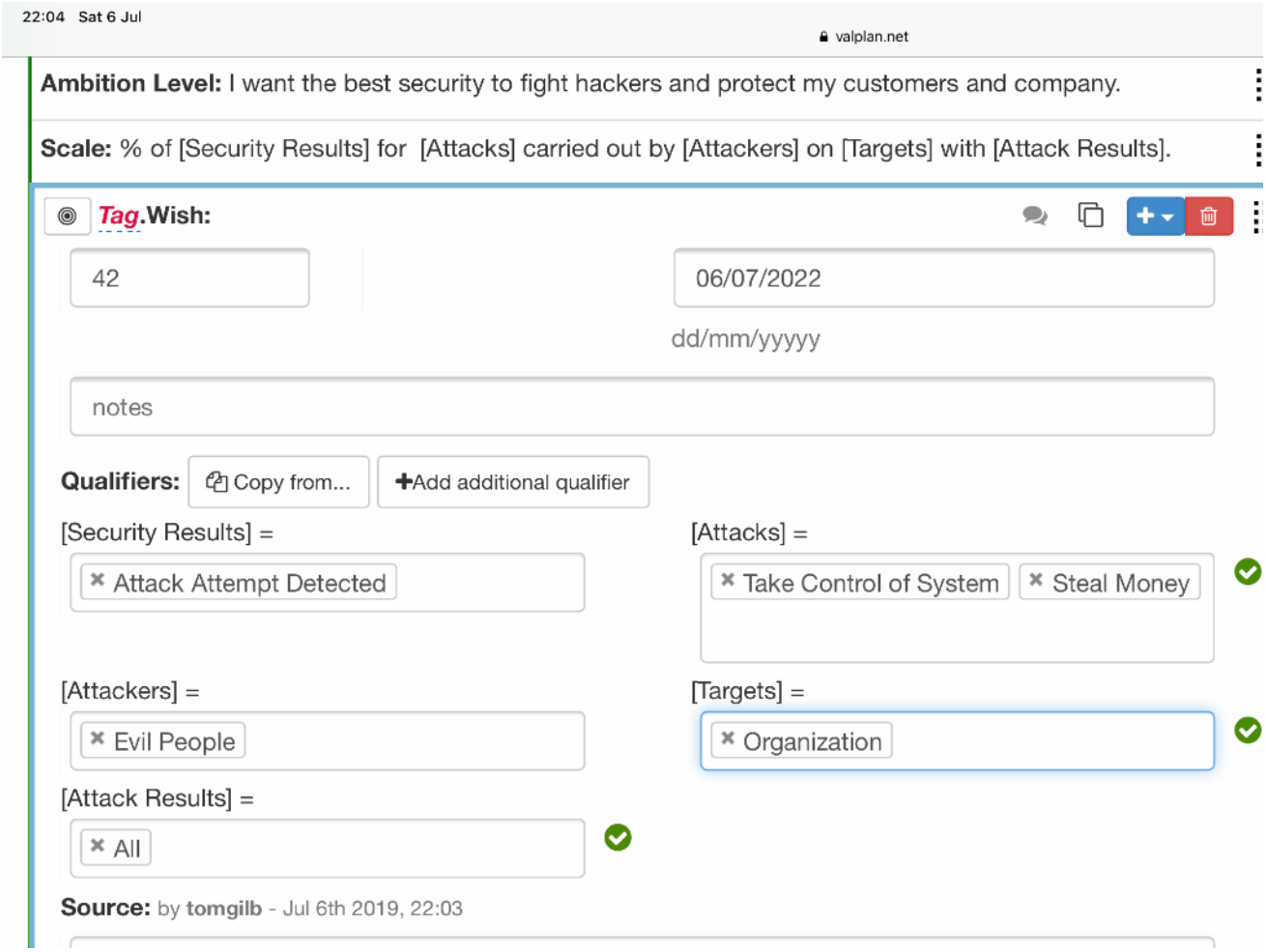


Figure 1.41 : after defining the Scale, I drafted my first Wish level. (Built on Fig. 1.40)

Note that it is a very small subset of all the Security Scale possibilities.

That is good. I can focus on this slice of the action, if it is high priority and critical.

I have a fair chance to understand it, and find security options and cost them.

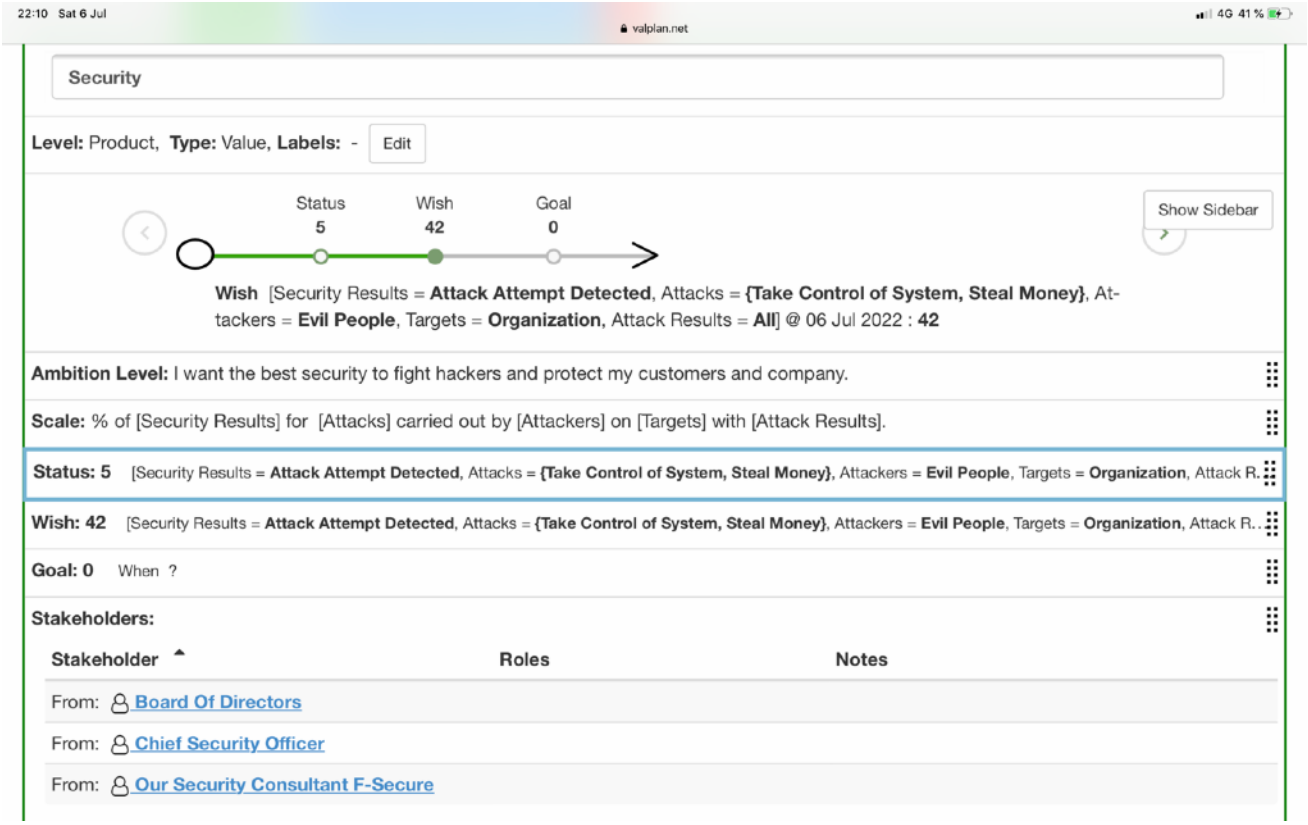
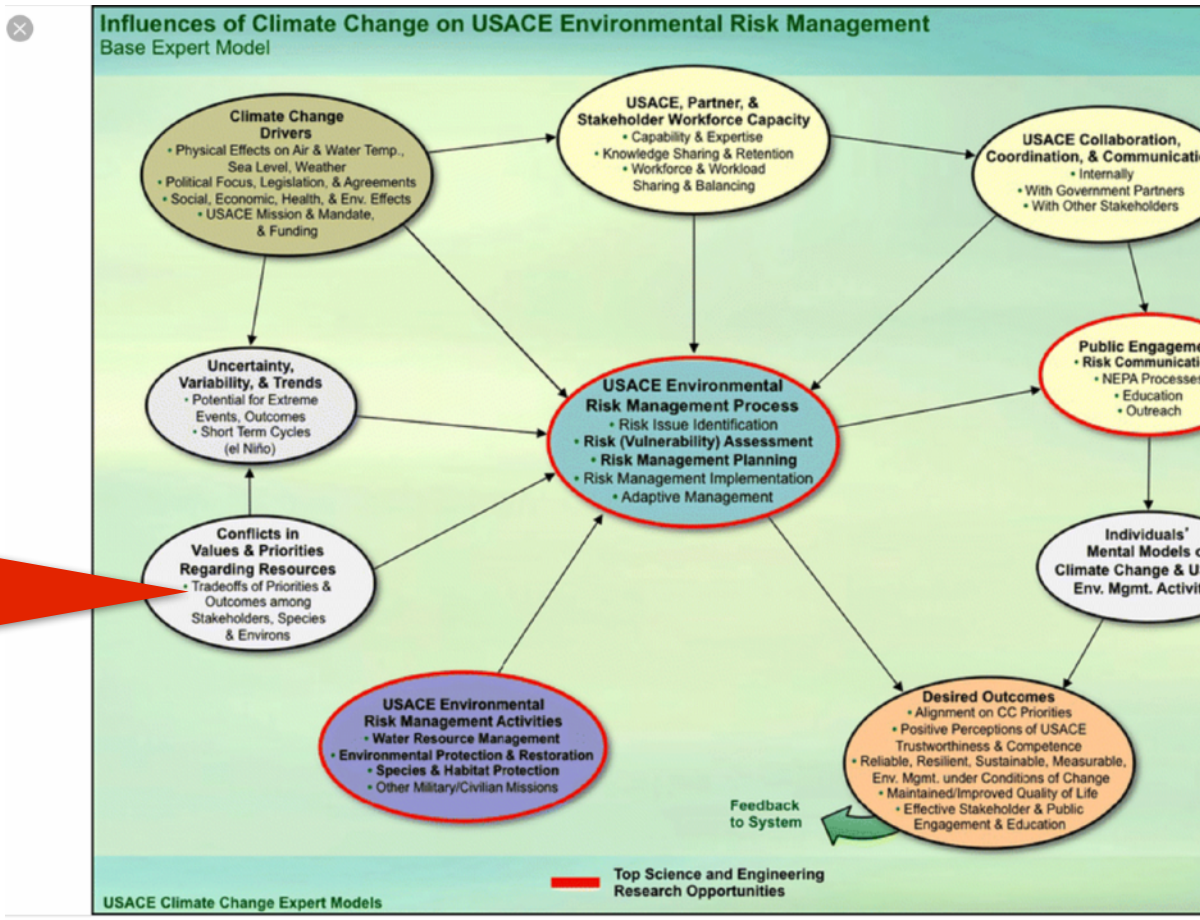


Figure 1.42 : So now I have my Wish specification. But I cannot possibly commit to a Goal (a 'firm committed promised value delivery from a funded project') because I have not identified and costed the necessary technology for delivering the Wish Level (42%) on time, 6 July 2022.

But at least the 'Wish' problem is much clearer.

We can understand, and discuss with our stakeholders on a much more realistic basis, than with the Ambition Level.



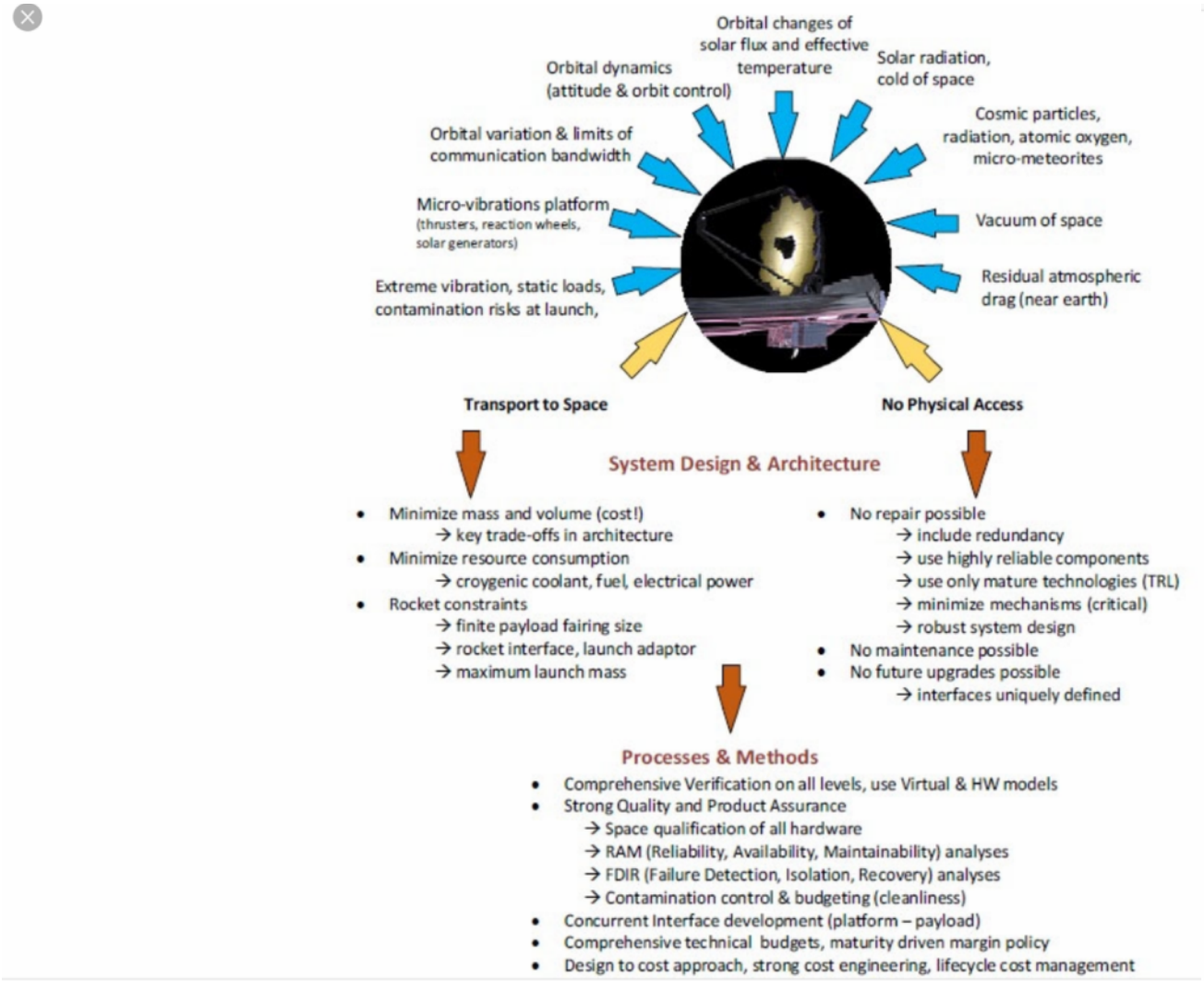
Springer

Climate change risk management: a Mental Modeling application

Figure 1.84 : a 'space' environment view of multiple Values, driving multiple design options with multiple constraints, to find satisfactory balance.

Figure 1.85 : With very many values, and very many international stakeholders, the trade-off process is in play, in a risk management context.

Figure 1.86 : here is a simple flow chart showing an iterative design and trade-off process, until satisfactory costs are reached.



SPIE Digital Library

A comparison of systems engineering challenges and practices

dipdlp.cedis.

Trade-off ar

12.7 The possibility of getting control over costs by Subcontracting

In the diagrams above, the possibility of getting control over costs, by evaluating cost-competitive suppliers is hinted at.

This can be done by direct bidding, contracting, competition, and asking *them* to do design-to-cost.[16]

[16] Agile Contracting for Results The Next Level of Agile Project Management: Gilb's Mythodology Column Agilerecord August 2013. concepts.gilb.com/dl581

But, perhaps the most important tool to do this with subcontracting, and really save costs is, to *use most of the advice about numeric value requirements in this book. To protect your project against cost reduction by means of undesirable value reduction.*

Anybody can cut costs, if they are not constrained by real measures of values and qualities expected.

In addition, this Value Delivery needs to be proven incrementally, rather than 'all at once at the bitter end'. Avoid big surprises.

Sub-contractor cost control:

- all quantified and specified Value Requirements are in the contract
- Payment is released when Values are achieved
- Work is done incrementally, so there is early and continuous proof of capability to deliver value for expected costs.
- Bonus for more-than-expected cost reductions.

Chapter 13. Change Control of Value Specifications.

Just because we quantify and structure Value requirements, does not mean they are chiseled in stone.

Change is inevitable and necessary. And quantified structured Value requirements are *ready* for systematic controlled change.

Here are some methods of managing change to Value Requirements.

13.1 The concept of a specification Owner.

A 'Specification Owner', or more precisely a Specification Object Owner is a person or group given sole power to change a specification object, such as a single Value Requirement.

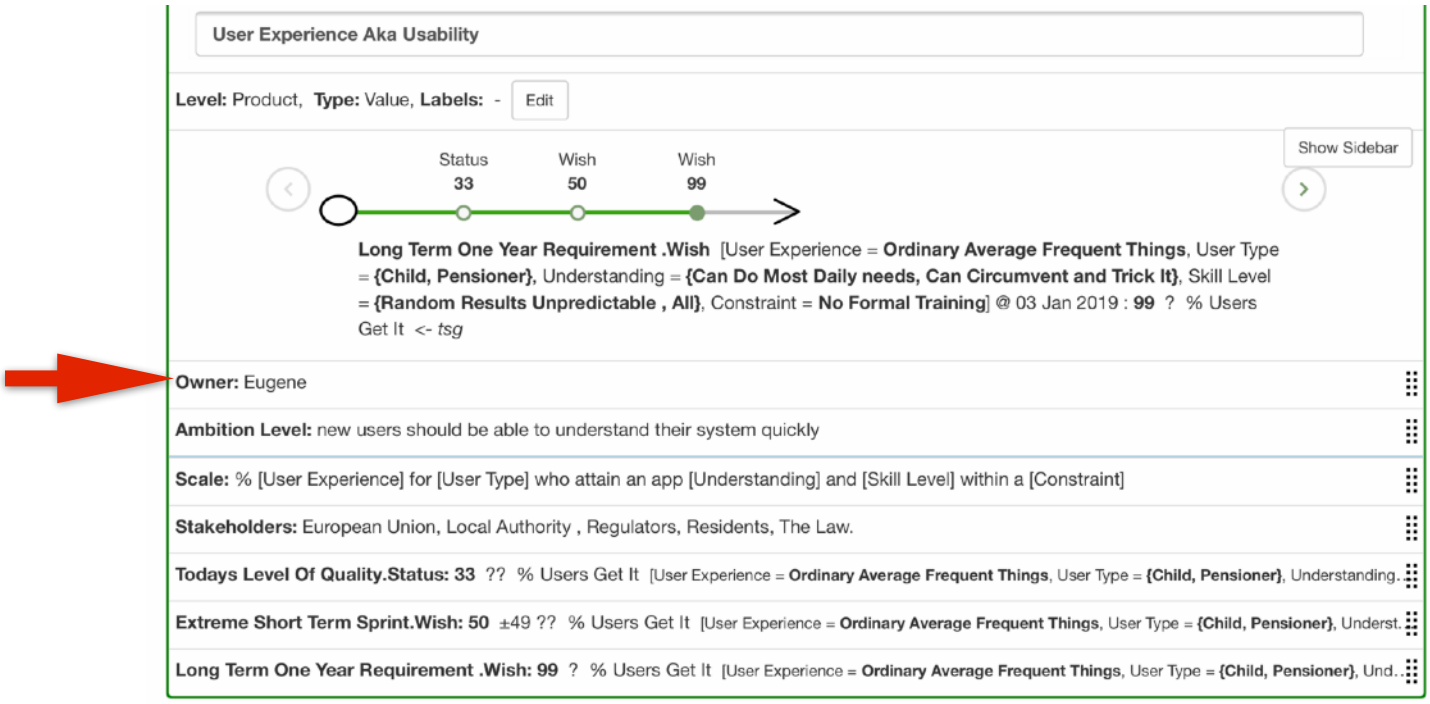


Figure 1.87 : 'Eugene' is the designated spec object (the User Experience Aka Usability Value requirement) Owner. 'Source BCS April 2018, Waste Management'

The spec Owner should:

- have accepted the Owner role voluntarily
- Be more than usually knowledgeable in the specific requirement
- Be interested in making the spec the best possible, over time; motivated.

The spec Owner is responsible for:

- receiving any hints from any sources, like stakeholders, of the need for corrections, updates, and changes
- Being password-enabled to actually do, and publish, any change
- Informing all instances, documented in the specification object, all relevant stakeholders, of the pending change, and the actual change (according to corporate guidelines for changes)
- Quality controlling, and reviewing changes, personally, or using others, and using Rules for specification best practices.

Notice what this means:

- we have *decentralized* change control to motivated people
- Control is no longer at a *committee* level, a level that does not really have time or interest in the many individual planning specification objects.

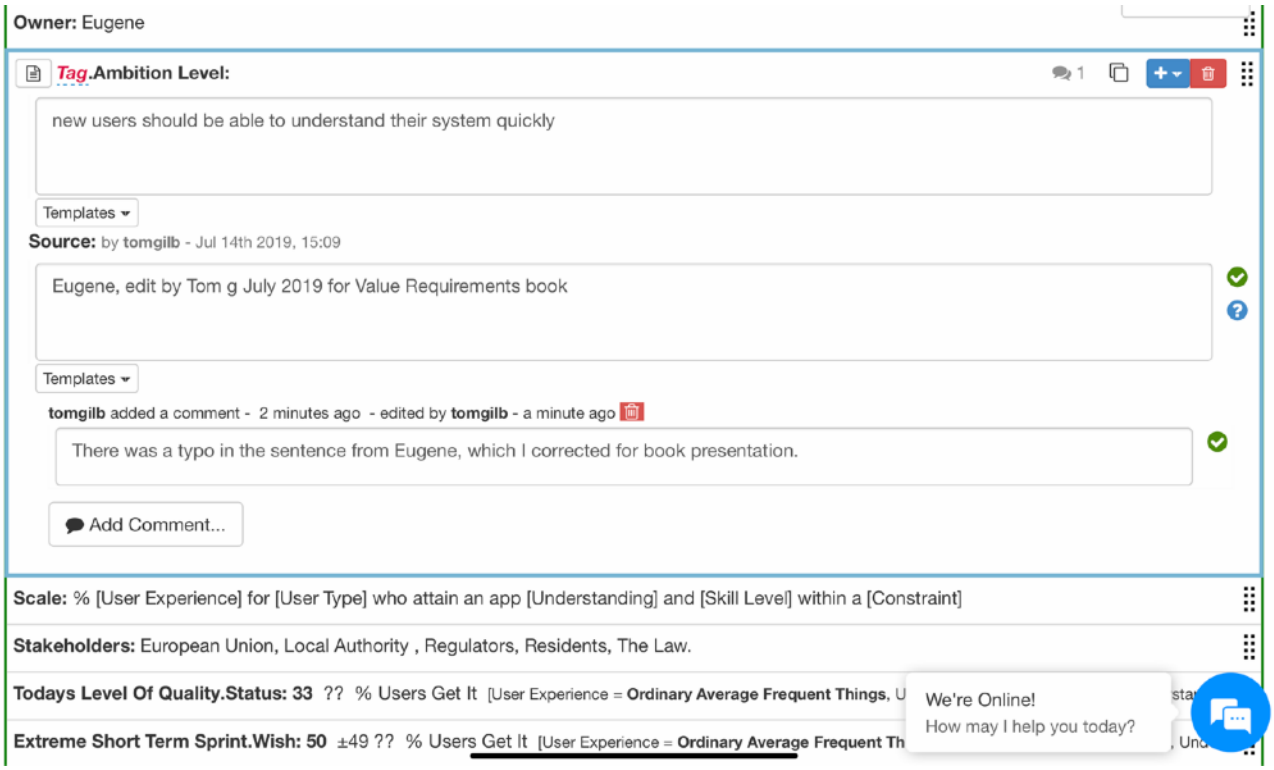
- You can use this decentralized responsibility to activate many people, including juniors and trainees, to grow in experience and motivation, into the larger planning system.

13.2 Annotation of change source, and time stamps

I am pretty clear that we need to annotate the 'Source' of each individual element of a plan. At the same time it is a good idea to get a time stamp for exactly when changes are made.

There are two change sources:

- the Spec Owner, or whoever actually keys in the change



- The information Source: 'who exactly said 64%?', or 'London?'

Figure 1.88: A detail window of the 'Ambition' parameter specification. Sources and change details are there.

A simple way of noting the source of any statement, is to use the keyed Icon '<-'

For example:

Wish: 99% <- Tom

Example 1S.

And then there is the question of exactly WHY a change was. Made, its justification, or background.

This justification is important because:

- We need to make sure the change is really justified.
- We need to explain to other stakeholders why the change is being made.
- Other stakeholders need to be able to argue about that justification.

A simple way of adding justification information can be:

Wish: 99% <- Tom

Rationale: this level is necessary to beat competitors.

Example 1T

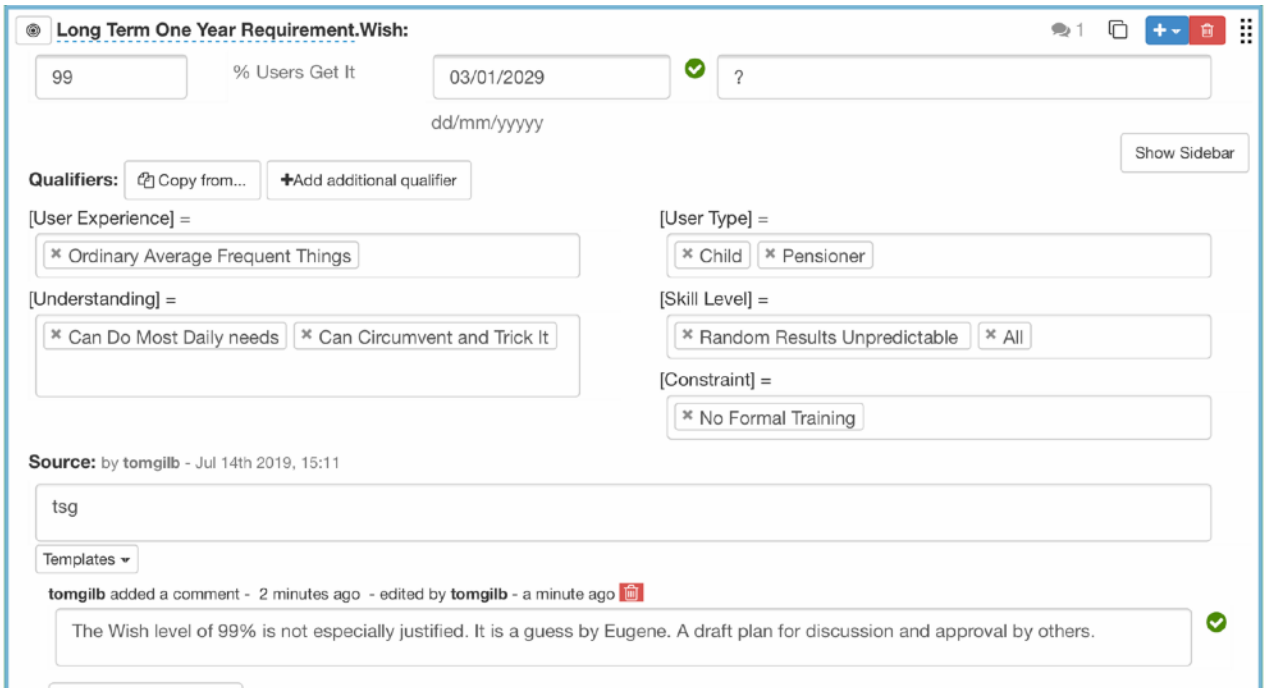
Figure 1.89: I used the Comment, in the Wish window to explain *why* the 99% level was set. An app can support and remind people to document 'Source' and 'Justify', in more detail

In this case the example is a Strawman, an initial draft subject for discussion, and improvement. Clearly not to be locked in and taken seriously, yet. Hopefully it is obvious to the reader why this is important to know, and know in writing, near the specification.

These *background* information can be backed up by specification Rules, like

1. The actual responsible Source of all critical specifications will be noted by personal name, position, or a group name.

2. All critical specification details shall be connected directly and locally to a justification, or Rationale, for why it is specified exactly they way it is. Even if the answer is that there is no good reason yet. It is a wild guess or strawman. Be explicit about that.



Example 1T

13.3 Ways of controlling the whole of the specification

How can we exercise control over the *entire* Value Requirements specification ?

There are a great many processes discussed here for ensuring the overall quality of the total set of Value requirements. And each component.

Examples of QC-Supporting Processes:

- Specifying 'background' things which allow us to verify and understand a specification (sources, stakeholders, justifications, comments)

- Giving power and responsibility to people, with *their name* on it (Owners, Sources)

- Leadership: showing that you *really* care to do things well, and knowing when not to overdo it, so it seems like a silly bureaucracy. A 'balance'.

- Retrospectives, root cause analysis, DPP Defect Prevention Process (ref. G) will all bring out reasons for problems. Hopefully 'root causes', including not yet taking the quality of requirements specs seriously enough. These analysis are your potential 'war stories' to remind people of the value. of 'doing it right the first time'.

- Motivation and culture change takes time, and leadership.

Chapter 14. A Review of Requirement Methods Compared to Planguage

14.1 General observations of methods for specifying Value Requirements

I am quite disappointed in the prevailing culture of dealing with Values, and Value requirements.

That is why I have had to invent my own way.

The current unhealthy requirements culture is very widespread, and new bad methods seem to spring up quickly and spread widely.

But our projects continue to fail, and part of that is bad requirements.

My central criticism is that most methods do not **quantify** the Value requirements at all. And the few that do so, do not do it *well*.

The following material, is for people who would like more-specific background.

They might have to attack some Holy Cows in their 'Temple', in order to deal with these problems.

14.2 A Checklist for understanding capabilities of value requirement Specifications

Here is a basic checklist, I do mentally, to **compare any requirements method with Planguage.**

1. Is the Value Quantified (or is it just nice words?). (“Highly efficient”)

2. Is a re-usable **Scale of measure defined well,** or is an oversimplified badly-defined scale only hinted at, together with the numeric level (“35% agree”)

3. Is the requirement **tagged** in some way, or is it just a bullet point, a sentence, or sub-clause?

4. Is there any systematic way used to **define terms** used in the spec, or are we left guessing at clarity and ambiguity?

5. Is there any **structure** in the Scale similar to our Scale Parameters? How is this variation and definitions of (whom when, why, where) dealt with?

6. Is there any way to annotate of capture the **justification** for a requirement?

7. How do they capture **sources** of requirements ideas?

8. Is there any set of **Rules** for requirement specification which could be the basic for Spec Quality Control: the defect level?

9. Is there any concept of **measured Defect Density**, which could give a basis for Exit from the requirements process?

10. Does the process simply capture a raw ambition level requirement, and leave it at that, or is there an attempt to analyze it and come up with a **better clearer requirement**.

11. Does the requirements process actually **permit 'designs' to sneak in as requirements**, when the real requirement is unstated, implied, or badly formulated? ('We want a password for Security')

12. Is there any concept of **stakeholders** for the requirements?

13. How good it the capture of **background information**, to help understand quality, risks, relations, priorities?

14. Are **Benchmark** levels systematically captured (Past, Status, Record, Trend)

15. Are the requirements suitable for **digital automation**? Can you program visual presentations, and analyze the specs?

16. Is there a **well defined classification and definition of requirements types**? (Function, Resource, Value, Mandatory Design, Constraint, Scalar Constraint, Scalar Target).

17. There is more, but this list should separate *strong* Value spec methods from weaker methods.

Chapter 15. SOME COMMENTS ON SPECIFIC METHODS

Not all these following methods below are 'requirements' methods, as such. But they are *related* to Value Requirements in interesting ways, and I want to share my observations with the reader, so that they themselves in turn, can argue with others better about the methods.

In some cases I have written a special paper in more depth about the method, and I shall refer to it for detail, and just give the 'highlights' here.

15.1 User Stories (ref. H, a)

I commented early in this book about User Stories. They are at the level of an Ambition Level, and we can use User Stories to start the process of deeper understanding of the implied Value requirement. But User Stories do not pretend to go into depth themselves.

My good friend Mike Cohn (Mr. User Stories) specifically referred to our Planguage methods, when asked on his website what to do about qualities and quantification.

As I said, I like the fact that the User Story does not merely have a 'requirement' idea, but that it specifically includes information about the 'stakeholder' (ok, 'User' only), and the justification (because)

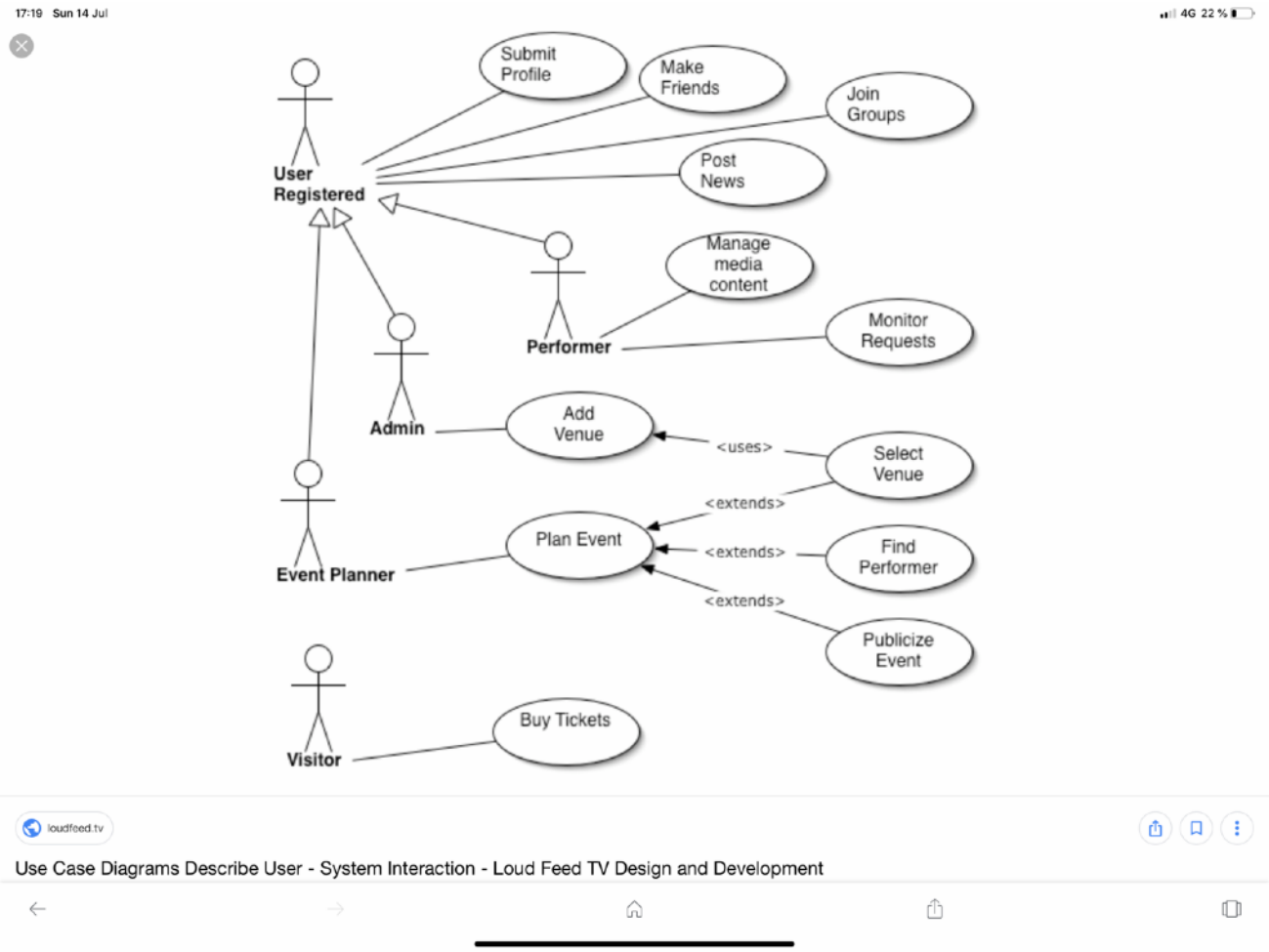
As a method for 'generating ideas about requirements', and possible values and qualities, for small and less-critical systems (no state of the art competition levels, no huge national health systems) user stories are quite OK.

My problem is, that I see user stories being used way beyond their 'level of competence', and I think user stories, as a primary requirements culture, are probably one initial cause of project failure.

Success and failure are not defined by user stories; they are more of a detail. But as we have pointed out earlier, the Value level 'Tolerable level' defines a failure border, and Goal level defines success.

User stories just do not deal with values and qualities, so we need something more, operating at at a higher level of controlling the system stakeholder results, values, and qualities.

My advice, if you are committed to using them, is to use them as an Ambition Level, a simplified departure point, and then analyze what the real, but implied-only, 'value level' has to be (derive a Scale and a Wish for example).



15.2 Use Cases

Figure 15.2.1: Use Case Diagram. Notice the 'Actors' (Admin etc) which we would prefer to make more general as 'Stakeholders'

Use cases are of course not complete requirements, nor Value requirements.

They are in fact very close, but not identical to, what I call ‘Scale Parameter Attributes’.

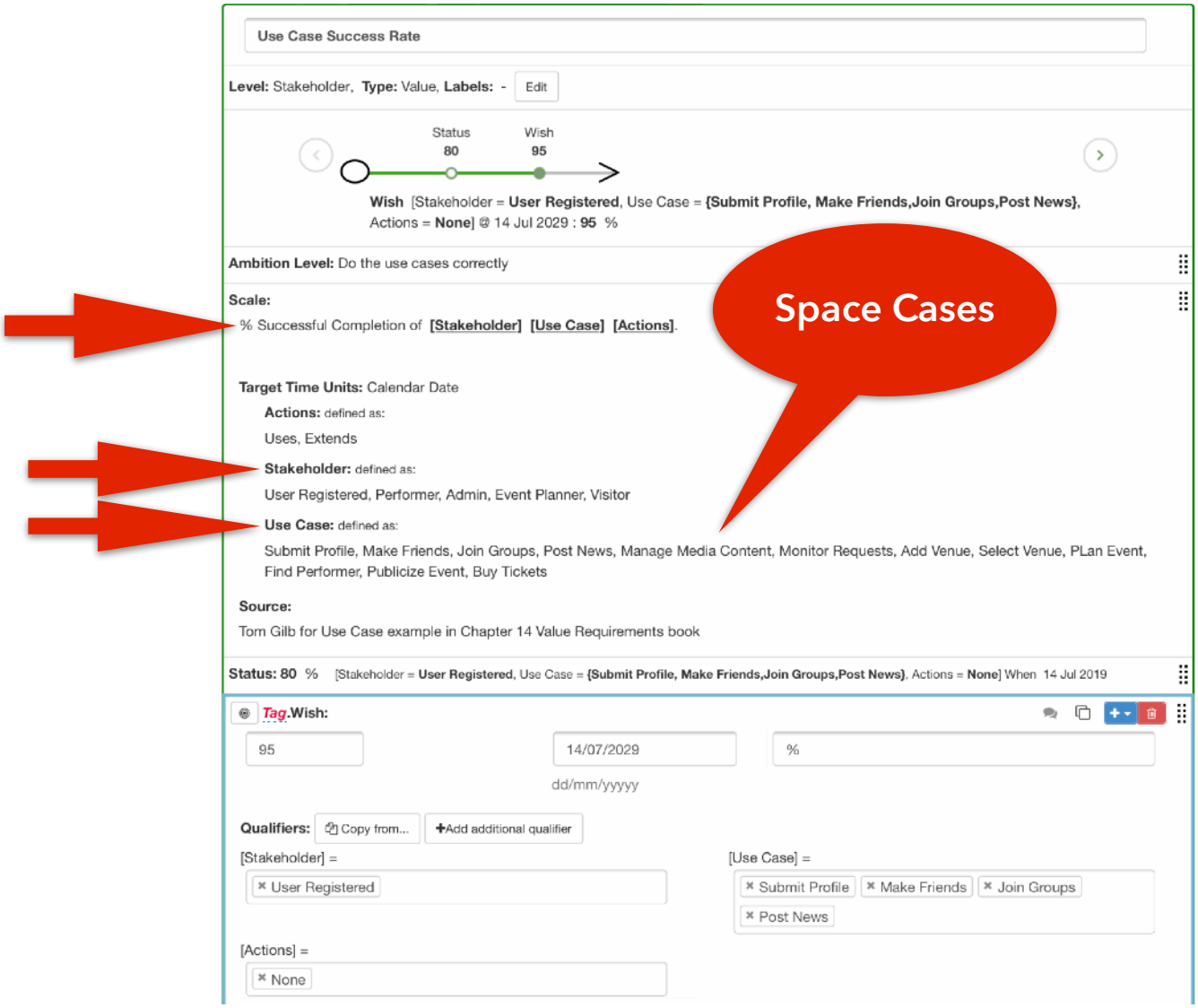


Figure 15.2.2: Compare this directly to the Use Case figure above. The 3 arrows point to Scale Parameters, each of which has 'Space Cases'.

So we can now more clearly see what Use Cases are. They are essentially Scale Parameter attributes, or for fun '*Space* Cases'.

So my 'Space Cases' (a term I just invented to express the broader scope than mere *Use* cases) are digitally integrated into the Requirement Spec., and can cover a broader *space* category.

For example we could add such Scale Parameters as:

- Places (where, city, country, area, groups like EU, NATO)
- Situations (War and Peace, Recession, Brexit, Natural Catastrophe)
- Experience and education levels of stakeholders
- Event Conditions (ordered, confirmed, attempted delivered, delivered, for example)

- And any other dimension spaces you need to express as conditions, for a requirement.

So my preference would be to not use the Use Case method, but instead to integrate the basic idea of Use Cases into Planguage with broader 'Use' Cases. In other words by using '[Scale Parameters]'. :)

15.3 Earned Value Management (EVM)

I recommend this EVM overview

https://en.m.wikipedia.org/wiki/Earned_value_management

I would have hoped that EVM would deliver exactly what the name implies. But it does not.

It does not deal with a set of critical Value requirements, at all.

It does assume Big Bang waterfall model pretty much, and 'value' is really just 'work done', or 'tasks', sometimes simply '% of budget spent' !

I recommend the blogs of a professional friend who spends his time fighting for non-corrupted, honest versions of EVM in US Government Projects, https://www.pb-ev.com. Paul Solomon, who has written a book on the subject with another friend that I have worked on several US Government Projects with, RalphYoung.

These guys are honest idealists, so you can trust what they say about EVM.

Of course when a desperate Government, *dictates* EVM, in an attempt to control greedy, and technically incompetent subcontractors, it gets used, and abused.

There is little EVM interest outside of those circles.

15.4 Functional Requirements and Non-functional Requirements

My definition of 'Function', and 'Functional Requirements' (which I call 'Function Requirements') is 'what a system does'.

Similar to the Use Case Actor actions. What they *do*.

But I observe that there is little agreed discipline in using the Function term. It can easily cover any type of requirements. And as often as not, 'function' can be applied to what really are 'design'. So the situation is messy.

A 'Function' can be programmed by a programmer. So can some designs. Both, functions and designs, are *binary*, present or absent. Nothing in between. Both are testable, for presence or absence. Both are in some sense, therefore, simple. 1:0.

People *outside* of IT do not seem to have a problem here.

When programmers were reminded that there were some qualities they were not good at, like *Usability*, they observed that this was 'not a function' or design they could program. So, they solved their dim understanding (of a Value) by calling it a 'non-functional' requirement or attribute.

I have seen what they then do with this requirement category. They specify it as 'TBD', to be determined, someday, when we figure out what it is.

One problem is that although people mean 'qualities' (and Values) when they say 'non-functional': things are not so simple.

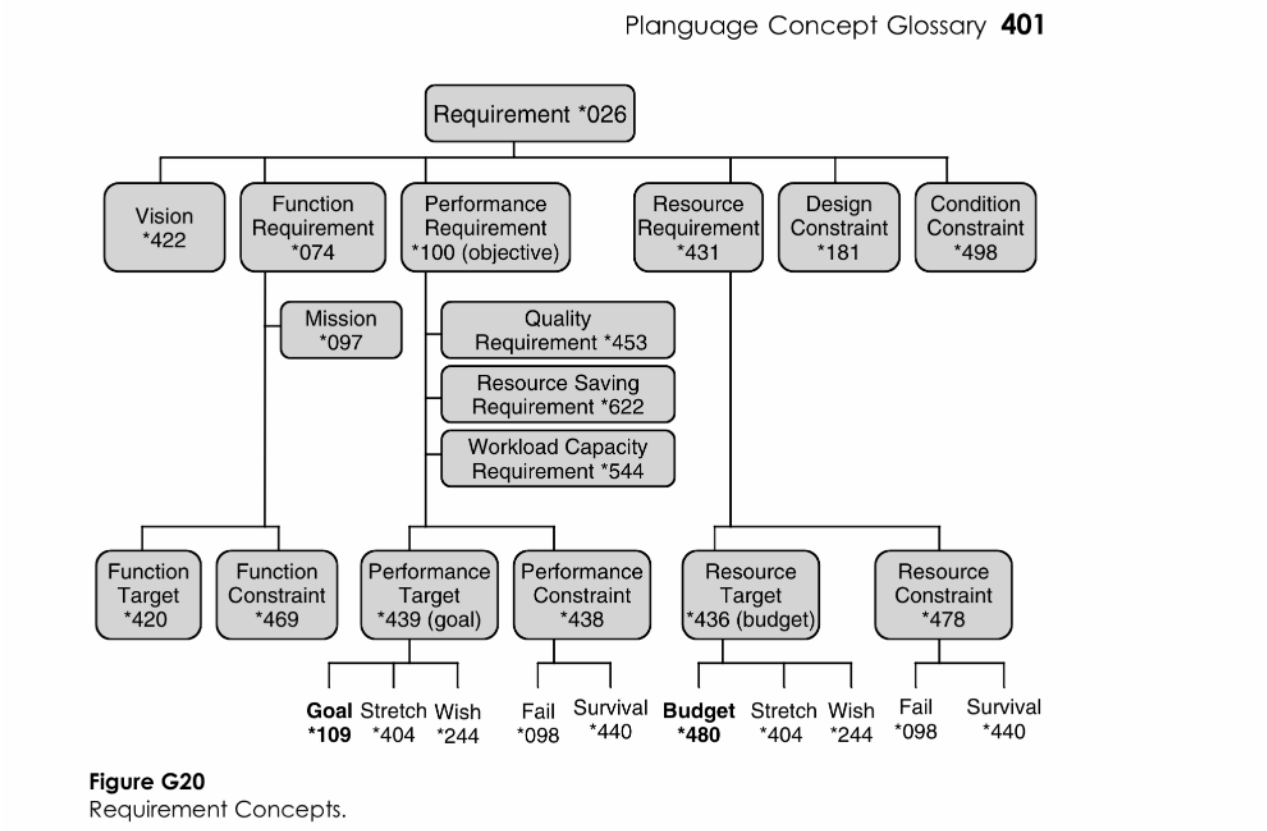


Figure G20
Requirement Concepts.

There are very many other requirement attributes to consider.

Figure 15.4.1: Planguage requirement concepts. From the 'Competitive Engineering' book. There is Function, then Quality, and all the others which are not functions!

From this 'Value Requirements' book point of view, we are interested in all those system attributes which stakeholders *value*.

That is pretty much everything, including Functions. That is why they are called requirements, I guess.

But this book has chosen to focus on the more-complicated *value requirement,* because it is variable, and people have a problem with variables. They don't stand still.

PS the use of the term ‘non-functional’ is a dead give-away that people have no real understanding of requirements.

15.5 Balanced Scorecard

Figure 15.5.1: a BSC with emphasis on Key Performance Indicators (KPI). This is the best of a lot of bad examples. It does try to quantify, and has benchmarks and targets.

Source: Datapine.com

The original Harvard Business School, 'Balanced Scorecard' failed in my opinion because it

- recognized there was an imbalance between financials quantification, and non-financials

- But it failed to quantify the non-financials (as a *norm*, not an exception)

Later efforts have tried to be better at quantifying the non-financials, such as the example above.

But (BSC improvement efforts):

- they still fail at tackling the really critical values
- They admittedly prioritize things, which they find easy to quantify (still 'unbalanced')
- They do not depart from the 'really critical values', and find a way to quantify them
- Notice in the example above, the total lack of qualities, or anything ending in in '-ility' ?
- They are avoiding the issue, because they do not know how to deal with it.
- Notice there are absolutely no product or service qualities in the example at all.
- Notice there are no well-developed Scales of measure at all, just highly-ambiguous ones. "Sustain Customer Retention". OK as an Ambition Level, but not as a well-defined Scale of measure.
- Some points for having both a Benchmark and a Target level. But notice no constraint, worst-acceptable-case level (Tolerable). No dates set on obtaining the target levels
- And notice a mystical "Likelihood of Reaching Target", done how? By whom? Any validation that it works?

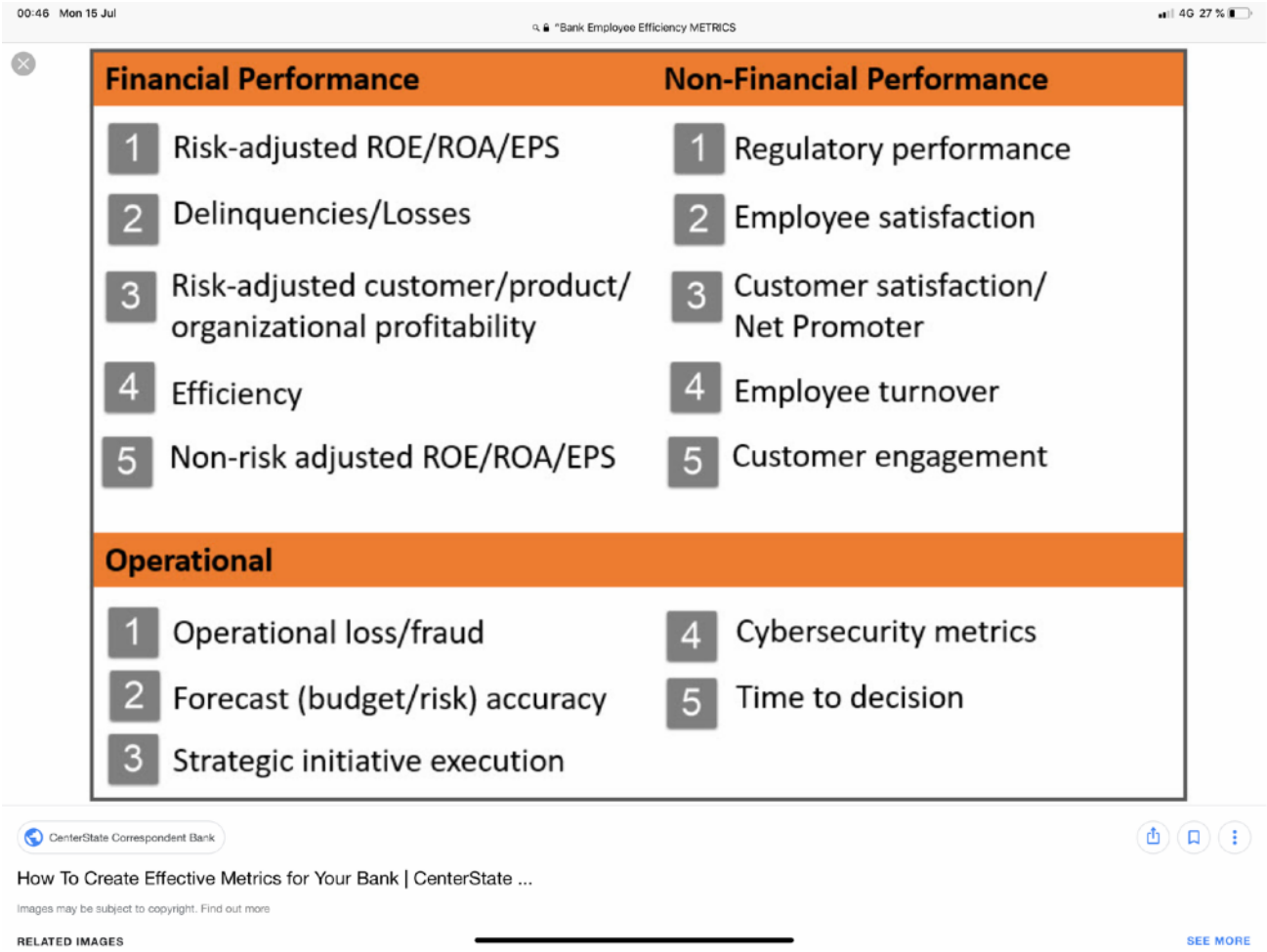


Figure 15.5.2: example of someone's idea of bank metrics in a BSC context. At this level they are just name Tags. I wonder what real Scales of Measure would look like?

Focused on finances? Here are 68 more financial KPIs your bank might want to measure.

Quality

9. **Client Survey Score:** Bank performance as measured by customer feedback. Many banks send out client surveys to gather performance-related feedback; tracking these responses with some type of internal scorecard is helpful. You can even create categories for response types (e.g. employee communication, variety of products/offers, speed of service, etc.) and track them individually, as well as your overall customer satisfaction score.
10. **Average Time To Close Issues:** Length of time from when a problem is identified to when it is solved. Issues may originate internally (operations, technology, etc.) or externally (customers).
11. **New Account Setup Error Rate:** The total number of new customer accounts created containing an error (e.g. typo or incorrect address, name, account type, etc.) divided by the total number of new customer accounts set up at the same point in time, shown as a percentage. This metric will ultimately link to the previous "Average Time To Close Issues" KPI.
12. **Accounts Opened With Insufficient Documentation:** The total number of new accounts opened with insufficient documentation divided by the total number of new accounts opened over the same period of time, shown as a percentage. This is similar to the previous KPI for banks, but in this case, the information is missing versus incorrect.

Figure 15.5.3: This example is getting closer to specified Scales of Measure.

Big US Government Bank Case (e)

I consulted with a large US bank, one that was key in the recession of 2008. The top management were trying to use Balanced Scorecard.

They were tearing their hair out in frustration to try to make it work. I saw their problem, and helped them solve it.

They were trying to communicate about a lot of 'soft' management objectives, that were not defined well*, no scales, and so everybody had to 'make up a definition' in their own mind.*

Their relief when I showed them how to do that, was immense.

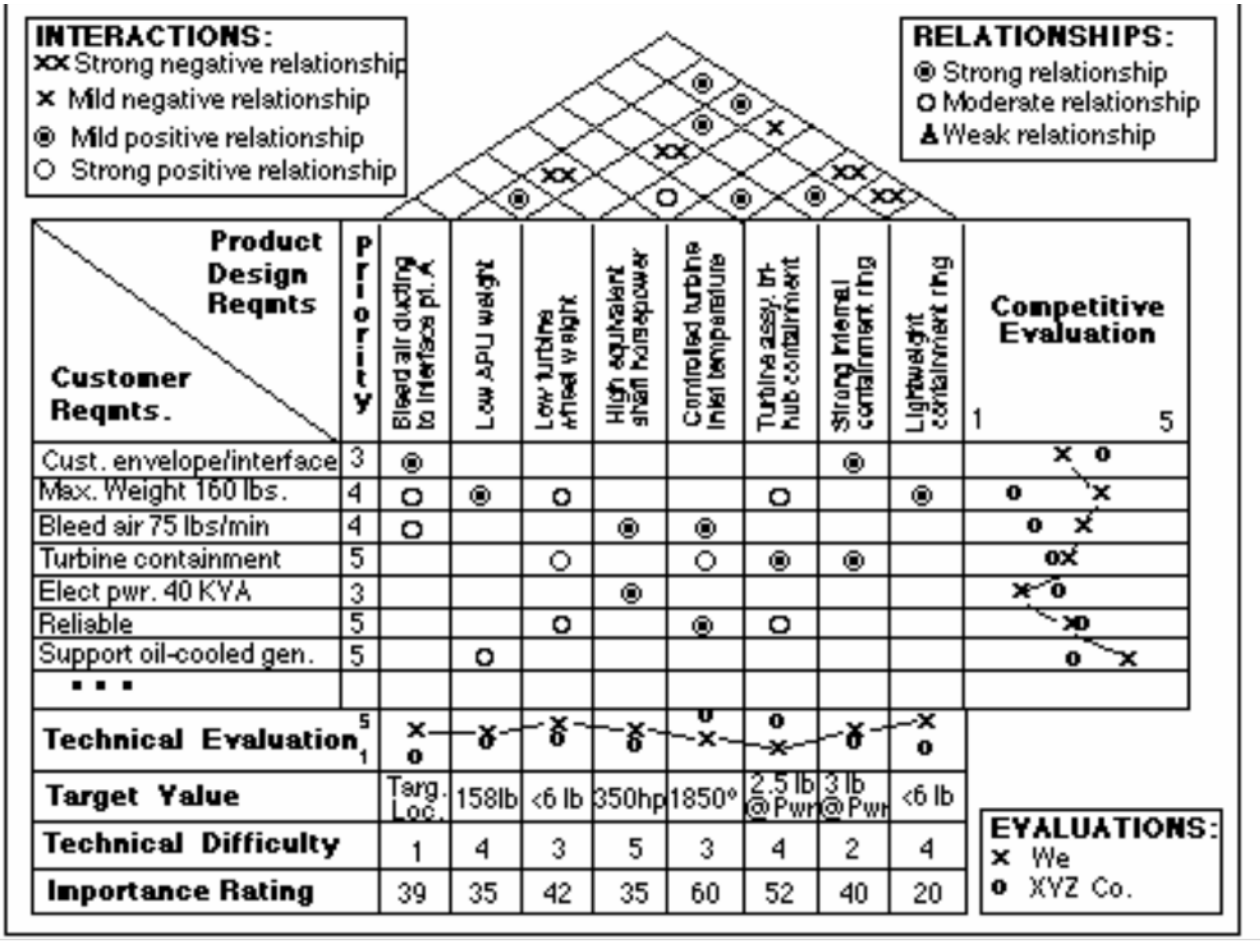
The details are in the reference (ref. e). But it is the old story, no knowledge or teachings yet, of how to define a concept with a scale.

This is not merely a BSC problem. It is a widespread cultural problem.

In this case it was at Harvard Business School that BSC was developed and published. But, I find that no business schools, which I can identify, teach managers the skills of this book: *how to define any Value Scale.*

Yet if we do as advised earlier, just search the internet for things like "Bank Employee Efficiency METRICS"

We get far more interesting metrics and insights than the BSC above. https://www.clearpointstrategy.com/bank-kpis/amp/



15.6 Quality Function Deployment (ref. I)

Figure15.6.1 : typical QFD and House of Quality example from www.

Just about everything is wrong with this method. I used to joke in my classes and lectures that QFD is so bad that it must be Japanese Fake News to destroy western industry. The fact that Toyota *really* did that intentionally, fake news to fool Western Industry, was revealed to me in 2018 (ref. f).

The above visual example is filled with badly-defined values, and subjective judgements. See the references (I, f) for details, and see the checklist above. But it *looks* so systematic!

The shocking thing from my point of view is that this is taught at universities, without any critical points against it being made.

The mentality, is, 'well they used it at Toyota, and Toyota make good cars, so it must be a good method'.

I'm told Toyota workers eat once a day at least, so that must be a good method for success too.

15.7 Togaf

Figure 15.7.1: Requirements are, formally, central in the Togaf Architecture Method.

There is much talk in Togaf of quantification, stakeholders, assumptions, constraints, KPIs, and Success definitions. But it is difficult to find any detail, of what this means in practice.

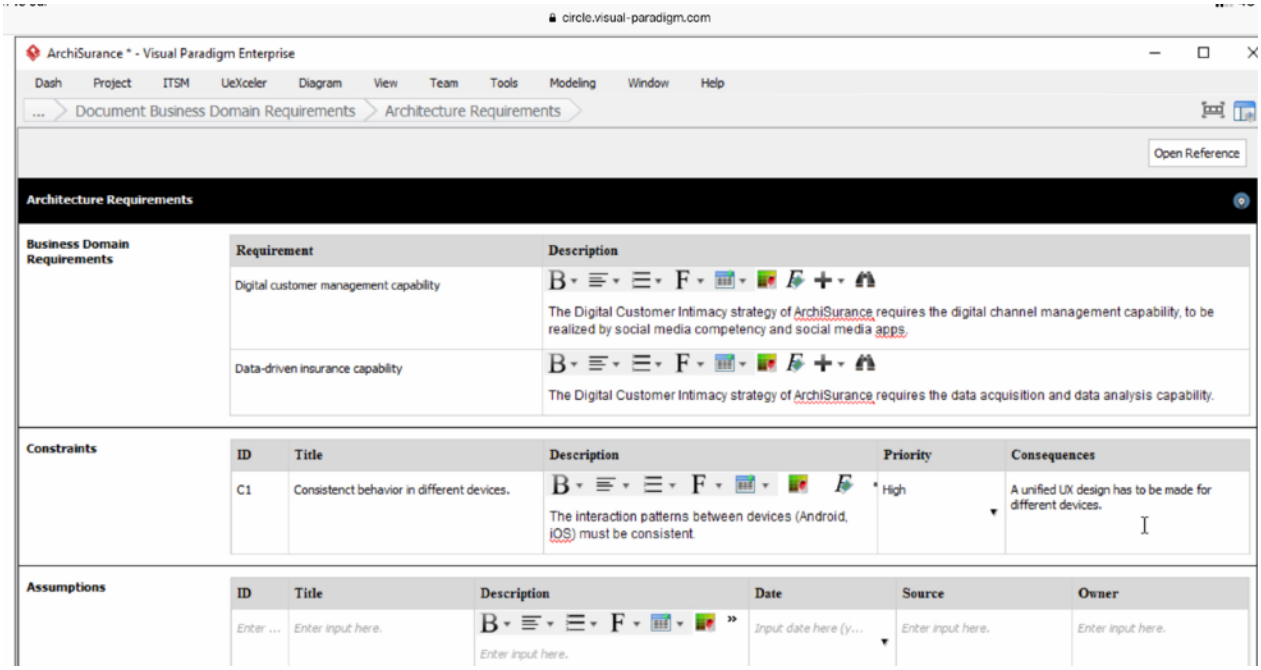


Figure 15.7.2 : Here is the best concrete example I could find, on the internet, of Togaf requirements practice. Just above this example, is text about '*quantified requirements*' being *mandatory*. Do you see *any* quantification?

I see some kind of an 'Ambition Level' ("Consistent Behavior", "...Capability") buried in the name tags.

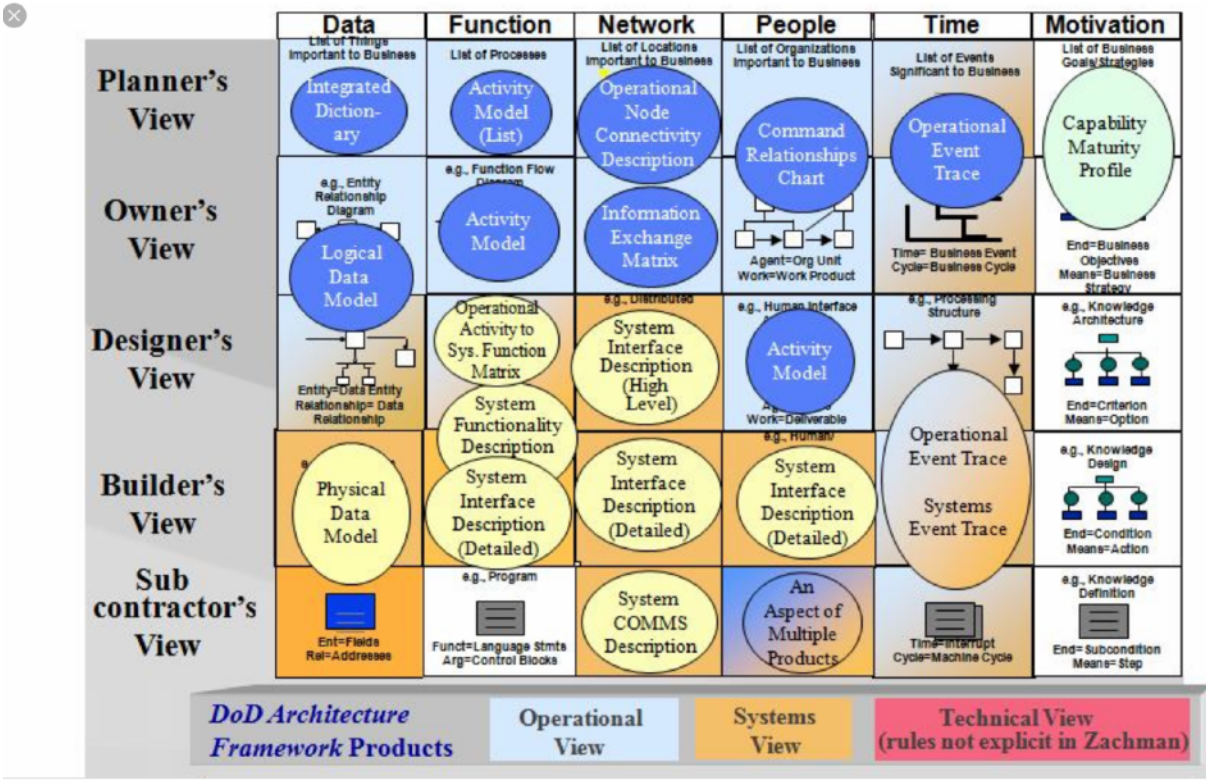
I also see the usual combination of a sort of requirement ("Digital Customer Management Capability") together with a suggested technical architecture ("the data acquisition and data analysis capability"). Bad combination: vague ends and suggested vague means!

I have also seen Togaf practice amongst my clients, and my Architecture Engineering course students. I was never impressed. Just disappointed in the practice.

My conclusion regarding requirements for Togaf is that the quantified practice they recommend, and I agree with, is simply not taught or practiced.

When architecture rests on such a bad 'requirements foundation', the result must be disappointing.

Togaf people are of course welcome to adopt the ideas in this book. These would conform with many of their stated ideals. No extra charge, just credit your sources.



en.wikipedia.org

File:DoD Products Map to the Zachman Framework Cells.jpg - Wikipedia

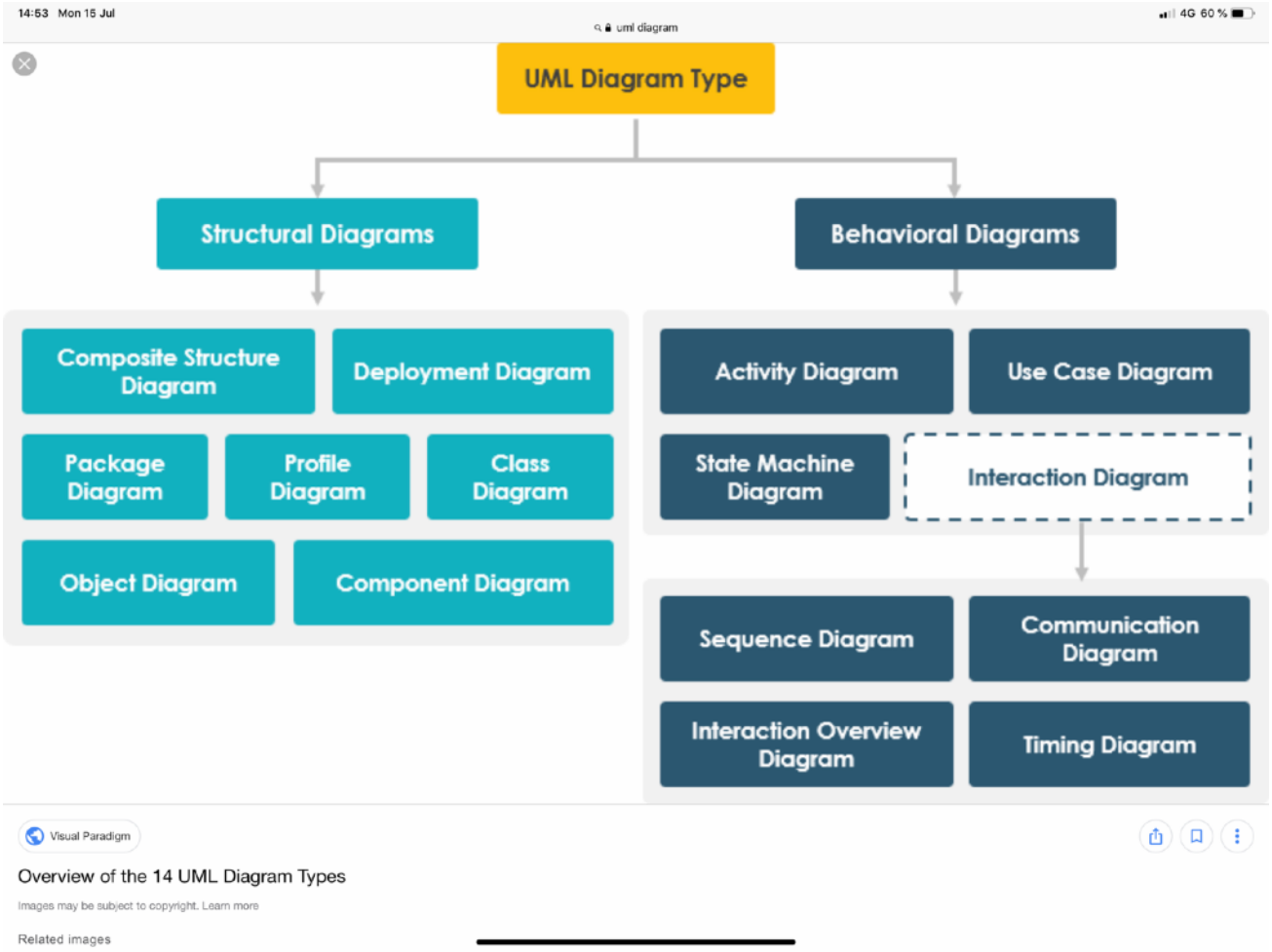
15.8 Zachmann Framework

Figure 15.8 : Zachmann framework mapped to US DoD Products.

I confess to a weakness for Zachmann. His framework covers a lot of bases, a lot of the system.

As this figure reminds us, we cannot expect much detail of a framework. It is up to the user of the framework to fill the intersections with specific methodology.

The Capability Maturity Model, level 4, was explicitly (Ron Radice IBM) based on my Software Metrics (1976) book, quantification of qualities and values. So that is an example of these Value Requirements ideas in this book, put in any framework (CMM, Ch. 15.14), you like.



15.9 UML: Unified Modeling Language

Figure 15.9: a set of UML modeling diagrams.

UML models a lot of things, but values, costs and qualities are not amongst them.

From my point of view this makes UML, and many others like it, quite inadequate for modeling the real world, and some of its most important aspects (Values, qualities, and costs).

I believe this is due to the built-in in narrow-mindedness of a computer-programming culture, where the program can be constructed without reference to costs and qualities.

15.10 Design Sprints (ref. g)

Figure 15.10.1 : Design Sprint 3.0

Design Sprints (1.0, 2.0, 3.0) are getting better, but they do not have any concept of quantification of Values as requirements. It is still a yellow-sticky culture, where the emphasis is on finding an app or web design, rather than departing from a clear set of multiple Value and constraint requirements. Maybe good for simpler problems: but I have looked and not found any studies comparing Design Sprints to anything else, for example in terms of project success, productivity, value for time spent.

Planguage offers a similar better startup week idea: The Project Startup Week (ref. K). It has been applied to large banking, aerospace, and defense projects successfully for decades.

15.11 The 'Evo' Project Startup Week (ref. K) : Values Driven Start

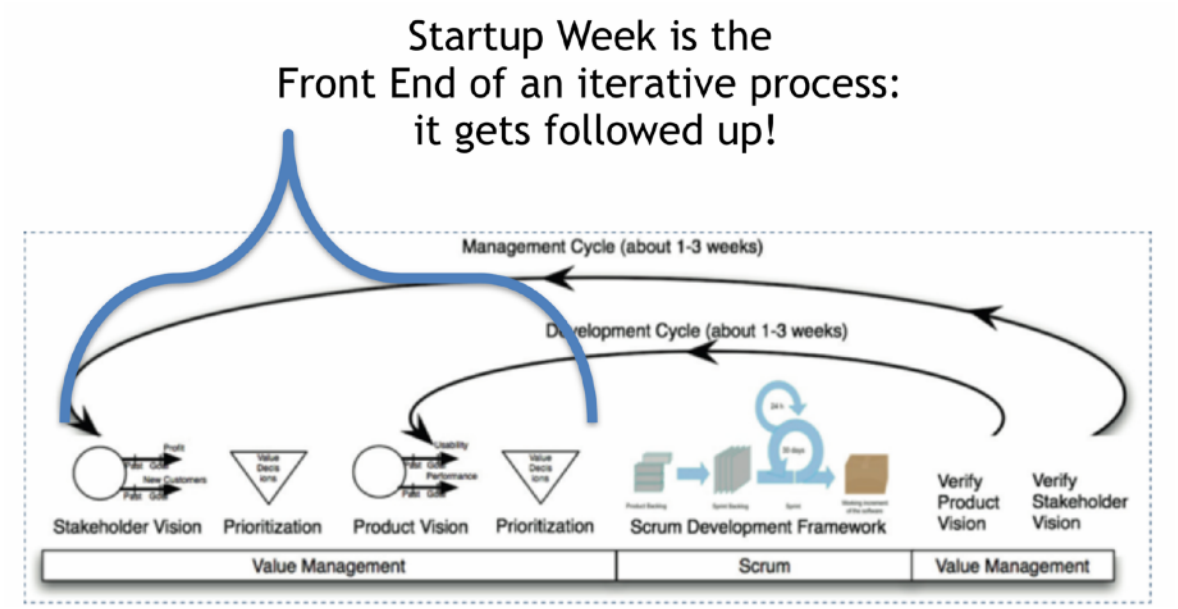
The Project Startup Week is fully compatible with the Value Requirements ideas in this book.

Day 1: Top 10 Critical Project Requirements Quantified

Day 2: top 10 architecture (design) options on the table

Day 3: Estimation of all designs impacts, on all Values and costs

Day 4: Decomposition of big designs into smaller ones, and selection of a high value-to-cost design-increment to implement in a' sprint' next week.



Repeat every week, Step 4, to increment towards the Value Goals.

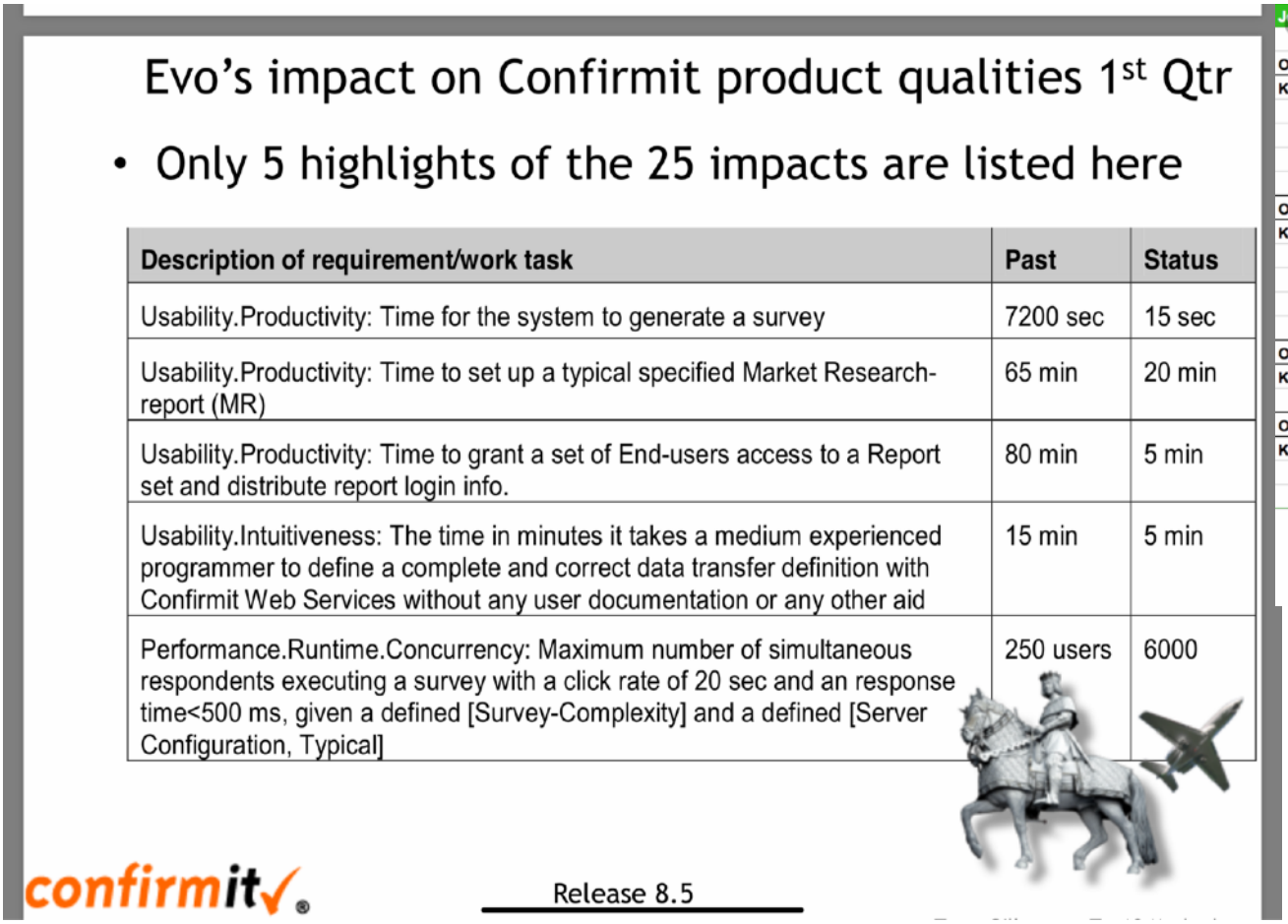
Figure 15.11.1 : The Evo Startup Week focuses on quick stakeholder value production, measurably.

This startup is primarily driven by a set of quantified critical stakeholder values.

It does not try to get a mock-up, or prototype, working in the first week. It tries to get real measurable results, a value stream, with currently existing products, services and systems.

It tries to learn by stakeholder feedback, and incremental measured results, what works, and what does not.

Figure 15.11.2 : The best 5, of 25, measurable Value improvements in 1st release, 12 weeks of increments, from. Start of using the Evo Incremental Value process, after a startup week to quantify the 25 Values most critical. Source, Confirmit.



15.12 OKR (K) Objectives and Key Results.

Figure 15.12.1 : an example of OKR planning. Notice the objectives are not quantified and clear ("Create an engaging newsletter"). The 'Key Results' are not business results, they are individual *tasks* (Interview 3 people"), which the individual *assumes* (hopes?) will produce the vague objective. Good luck!

I have no problem with OKR as a way to make individuals and small groups plan their weekly work tasks. Maybe it is a good thing?

But I have had problems finding any studies of OKR, and even good case studies, which *illuminate the values we get for the costs.* What was the result at Intel? Do they still use OKR?[17]

[17] From Erik Simmons, July 2019: "During my tenure, OKRs were still used, but many teams had shifted to (or added) Landing Zones, which had elements of Planguage (and our training recommended using full Planguage behind each LZ row for clarity). Some teams still used OKRs to drive time-based behavior at the quarterly or yearly level, perhaps out of cultural inertia (though that was relatively low at Intel overall)."

OKR is in no way a replacement for Value Requirements, which operate on larger systems (products, organizations, services) and guides us to find ways to create measurable value in the short and long term.

For clarity, I do not think that our Value Requirements methods are appropriate for this level of individual task planning.[18]

I would like to think that these same individuals, are all part of some larger projects, and that they are interested in, and committed to, improving Value requirement levels.

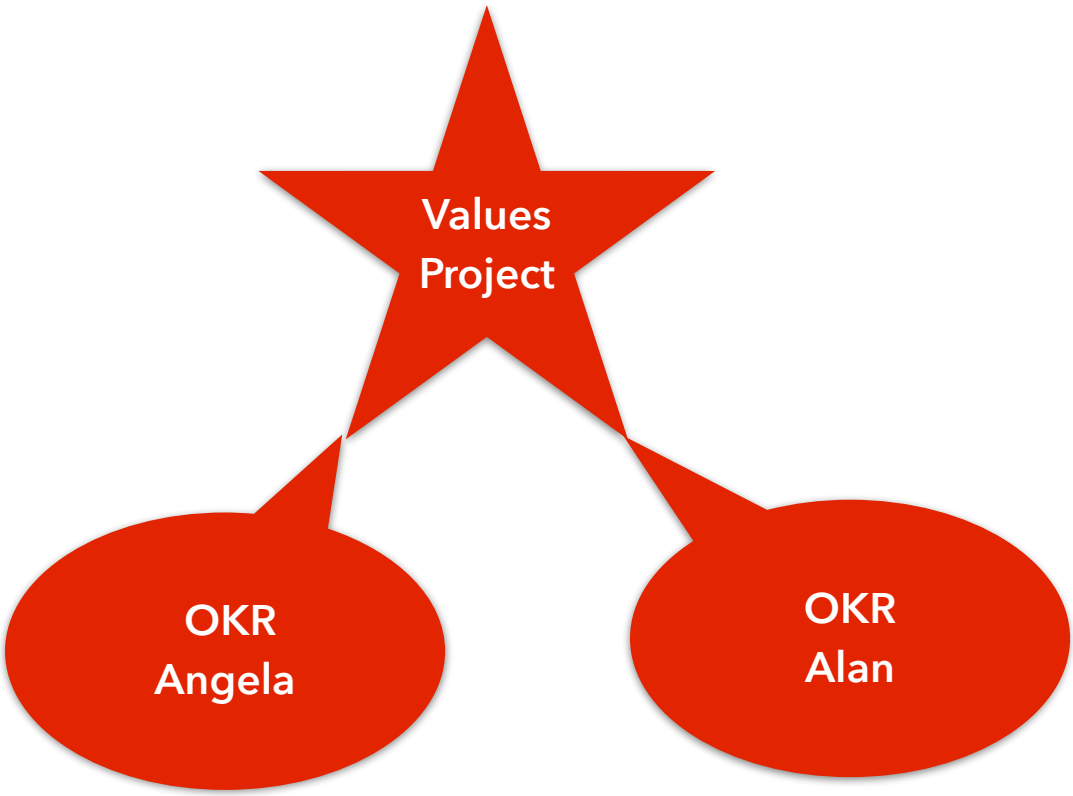
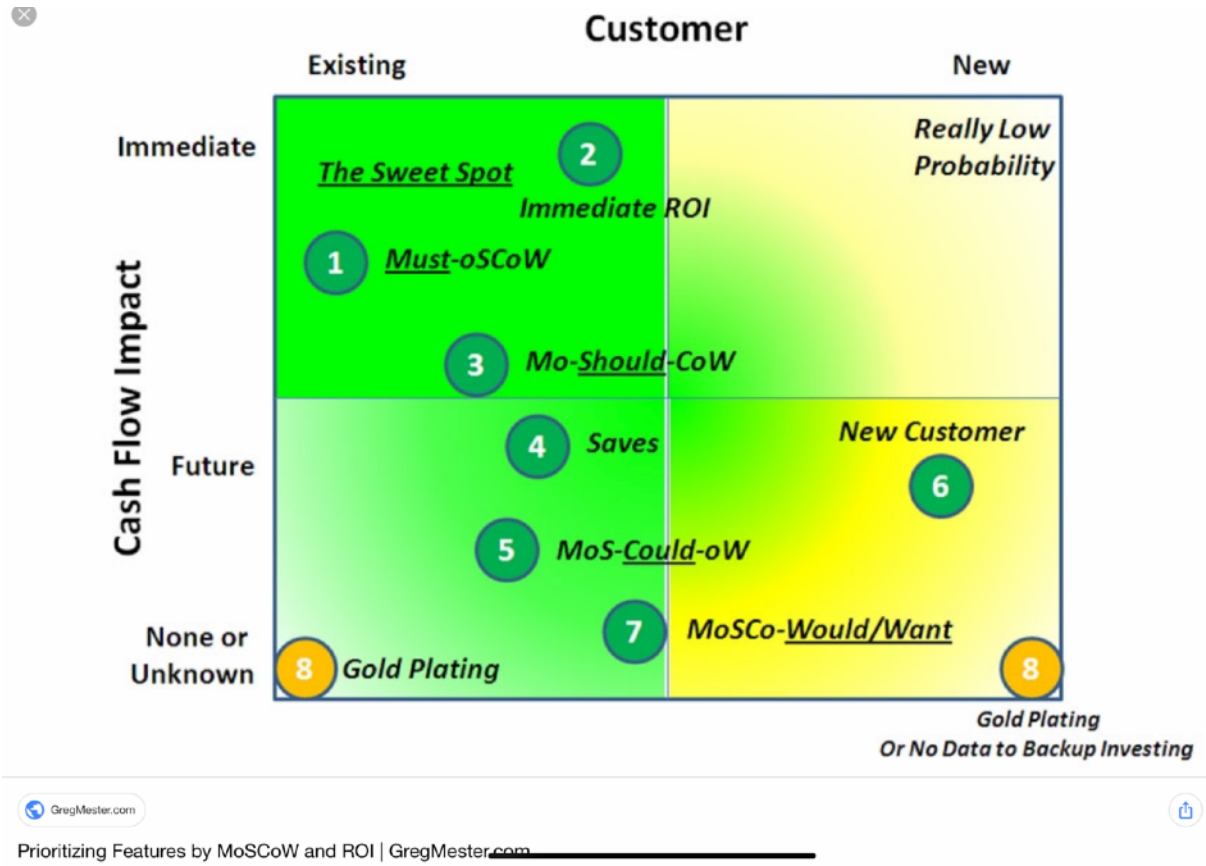


Figure 15.12.2: Individuals and small teams need to be well aligned to a higher 'project' level set of quantified Value requirements. OKR might help individuals with tasks, but it has not been designed and practiced to directly align with a higher purpose. This could well be because the higher purpose (Value Requirements) was never well-defined at all: as most of the methods in this part of the book fail to do.

[18] our tool ValPlan.net does support task planning.



15.13 MoSCoW: Prioritization Method.

Figure 15.13: a presentation of the MoSCoW prioritization method, which tries to bring in financial and market factors in the decision-making.

Prioritization of actions is necessary when resources are limited, as they always are.

I am not impressed with most well-known prioritization methods, this one included, and especially fixed-weighting methods, as are found in for example Balanced Scorecard, and Quality Function Deployment (see above BSC, QFD).

But I'll admit that bad prioritization methods are better than none.

Here are some points about prioritization (ref. E)

Methods for simple short-term prioritization might need improvement for large, complex, critical projects

There are a variety of different **resources** that might be considered in priority decisions (time, staff, capital cost, reputation, hardware capacity, and many more). Even combinations of resources might be considered.

We need to be clear about what is being prioritized. Most methods seem to assume it is features, functions or User Stories: all of which are a bad idea.

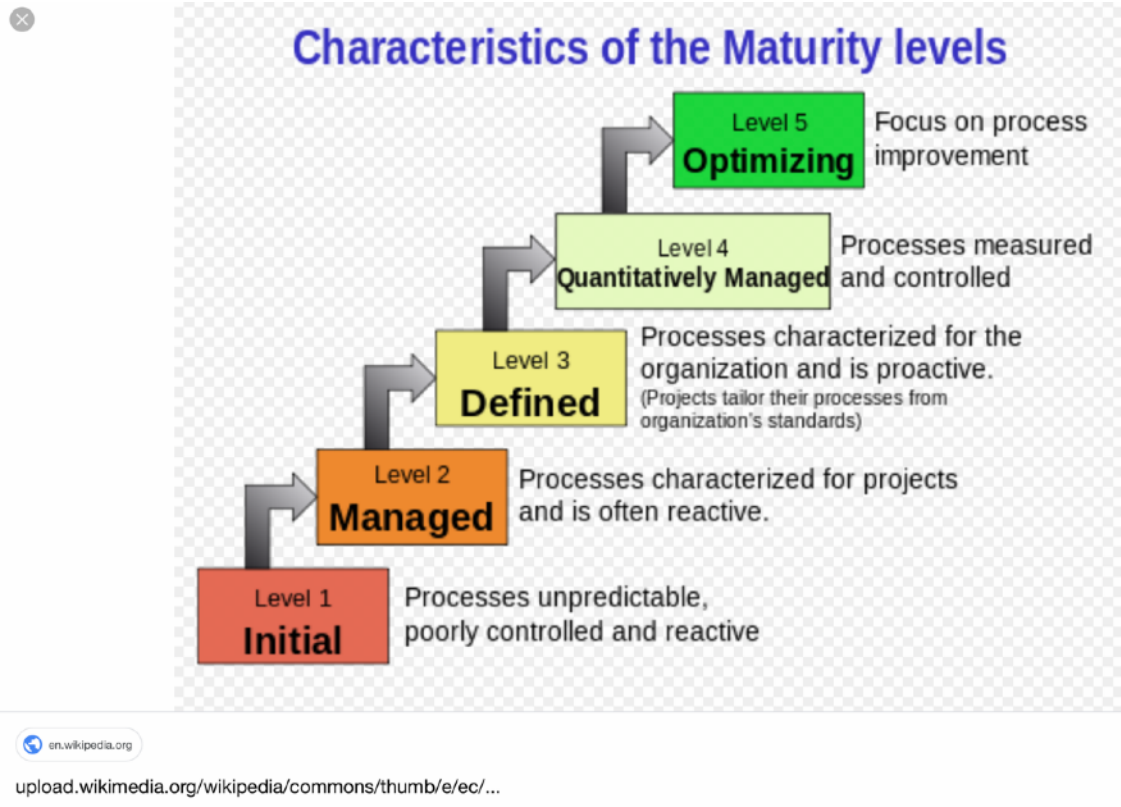
I think if *stakeholder critical values* are the main point of all projects, then we need to prioritize getting to the Value requirements, and stop when we get to one. Then re-allocate resources to reach other value target levels which are not yet satisfied.

I think asking a single stakeholder, or a Product Owner to choose a priority is a bad idea (but better than none) because they do not have an overview of the rest of the stakeholders needs, and the resource situation in the future for the project.

In short I think priority needs to be computed logically, with delivery-step by delivery-step, based on an overview of the critical decision factors.

MoSCoW is for projects that do not understand or plan critical values quantitatively. It is not for large or complex projects.

It is path to failure I would guess.



15.14 CMM: CMMI, Capability Maturity Model

Figure 15.14: CMM is a tool for assessing the maturity of an organization and its working processes. [19]

PS Good suppliers should get well paid for **value** delivery, incrementally.

Forget 'Maturity Levels', for organizations.

Focus on Value, performance now, let organizations figure out what to do themselves (their cost-effective *processes*) to get paid: The Free Market of Competence, I'd call it.

[19] The CMM level 4, 'quantitatively managed' is mainly based on my 1976 Software Metrics book ideas, according to Ron Radice, who invented it at IBM. But I think the focus on development processes, rather than stakeholder value, is a mistake. We need stakeholder value first, and *process* cost-effectiveness second.

Chapter 20. Summary of Value Requirements

Value Requirements are the main reasons for projects.

Other requirements are just useful and necessary details, to be taken into consideration.

We need to clarify and quantify the Value requirements just to have a fair chance of success in delivering them.

Then you have to consider many 'conflicting' Value Requirements at the same time. Finding reasonable balance. Extremes destroy the whole.

Then you have to consider all other types of requirements, like the functions, legal constraints and resource budgets.

This is getting kind of complex, isn't it?

But you have to do it professionally, or you will fail too early and too often in delivering the success factors: the improved Value Levels.

Figure 20: Values are many, and complex to co-ordinate. But Values must also consider all other types of simultaneous requirements.

Clarity of specification is the first line of attack on this problem.

Appendix 21 Book References

(1) CE: **Competitive Engineering**, 2005, Tom Gilb

Get a free e-copy of 'Competitive Engineering' book.
https://www.gilb.com/p/competitive-engineering

https://www.amazon.com/Competitive-Engineering-Handbook-Requirements-Planguage/dp/0750665076

(2) SM 1976. **Software Metrics**. ISBN-13 978-0862380342. Library or used copies only.

(3) VP 2016. **Value Planning.** Tom Gilb,

"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.

(4) VE 2017. **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.
This is the core front end of the Value Planning book (3).

(5) LD. 2018. **Life Design.** - eBook https://www.gilb.com/offers/JHHzGSER/checkout

(6) CC 2018. **Clear Communication**. https://www.gilb.com/offers/Y36JRL6g/checkout

(7) IC 2018. **Innovative Creativity.** https://www.gilb.com/offers/FnExtaw9/checkout

(8) PPP 2018. **100 Project Planning Principles**. https://www.gilb.com/offers/Shju4Zqn/checkout

(9) **Technoscopes** 2018. https://www.gilb.com/offers/YYAMFQBH/checkout

(10) PoSEM 1988. **Principles of Software Engineering Management.** https://www.amazon.com/Principles-Software-Engineering-Management-Gilb/dp/0201192462. $46

(11) SI. 1993. Gilb & Graham. **Software Inspection**. https://www.amazon.com/Software-Inspection-Tom-Gilb/dp/0201631814

(12) **Value Design**, 2019. See leanpub.com/ValueDesign

(13) **Value Management**, 2019. See leanpub.com/ValueManagement

(14) **Sustainability Planning** , 2019. See leanpub.com/SustainabilityPlanning

(15) **Value Agile,** 2019. See leanpub.com/ValueAgile

(16) Value Requirements (this book. leanpub.com/ValueRequirements

Appendix 22. Papers References, with Free URL Links

(A) **Intel** Cases. Simmons, Terzakis

J. Terzakis,
"The impact of requirements on software quality across three product generations,"
2013 21st IEEE International Requirements Engineering Conference (RE), Rio de Janeiro, 2013, pp. 284-289. I think you have the full paper, but let me know if not and I can resend. There's a lot of good background there.
https://www.thinkmind.org/download.php?articleid=iccgi_2013_3_10_10012

FREE LINK:
(with Gilb Annotations) https://www.dropbox.com/sh/cs9hke3uvgg4gp3/AACadHeI95lZpHzVqGKXSXDra?dl=0

PAID LINK 2013 RIO PAPER
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6636731&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6636731
Paper link requires purchase and signin

(B) **The Top 10 Critical Requirements are the Most Agile Way to Run Agile Projects**

http://www.gilb.com/dl554

(C) Confirmit Test. Try to get John Watkins analysis of them, a free paper not his book

https://www.cambridge.org/core/books/testing-it/confirmit/275AE17A5603289AB1F129A418572E1C

(D)

Confirmit Paper Gilb and Johansson
From Waterfall to Evo

http://concepts.gilb.com/dl32

(E) **My Deeper Priority Writings**

1. **Managing Priorities, A Key to Systematic Decision-making. With Mark Maier, 2005 (paper)**

http://www.gilb.com/DL60

2. **'Choice and Priority Using Planguage:**
A wide variety of specification devices and analytical tools'. (paper)
http://www.gilb.com/DL48

3. **VP Book, Chapter 6 Prioritization**
https://www.dropbox.com/sh/34llx1a7ckyagxl/AAA0pDzSxN5WmoP9lOKR0Mpca?dl=0

(F) Gilb '**Estimation or Control**' paper
SQP Magazine, USA
http://www.gilb.com/DL460

Slides made for BCS SPA June 1 2011
'**Estimation, a Waste of Time'**
http://www.gilb.com/DL70

(G)
Defect Prevention Process
Mays and Jones IBM SJ paper on Experiences
http://agileconsortium.pbworks.com/w/file/fetch/1527643/Mays1990ExperiencesDefectPreventionIBMSysJ.pdf

See also 2 DPP Chapters in Gilb & Graham
Software Inspection, 1993

(H) **"User Stories: A Skeptical View"**
http://www.gilb.com/DL461
User Stories paper by Tom and Kai Gilb
In Gilbs' Mythodology Column, Agilerecord.com March 2011
www.agilerecord.com/agilerecord_06.pdf (whole issue)

(I)Gilb and Brodie, **'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

(J) **OKR Objectives and Key Results: what's wrong and how to fix it.**
http://concepts.gilb.com/dl879

Paper 2 Feb 2017

(K) **Project Startup Week**
Agile Project Startup Week Paper in
Gilb's Mythodologies series
gilb.com/dl568

And

‘An Agile Project Startup Week’
91 slides pdf
Talk slides pdf from ACCU Conference April 9 2014
90 minutes talk
Includes Startup Planning for Business Startups, Confirmit, US DoD case, 2 Bank cases,
Detailed Startup week outlines and links to sources.
Bristol ACCU Conference
http://www.gilb.com/dl812

(L) **Principles of Systems Environments**
http://concepts.gilb.com/dl961
Gilb slides based on Pawel Nowak paper at GilbFest 260619

Based on Pawel Nowak, NOWY, “Context - between model and reality. My attempts to catch the elusive notion”. Talk at GilbFest, London, June 26 2019

(M).

Appendix 23. Slides and Talk Video References, with Free URL Links

(a) US: **User Stories as value requirements**. Slides NEED TO MAKE ONE OR IN BOOK

User Stories with Value Metrics 20Feb17.pdf http://concepts.gilb.com/dl883

Slides Based on Kai Gilb's Experiments
Mike Cohn commented he liked this.
See also reference to paper (H).

(b)
tinyurl.com/GilbTedx
link tested Sept 2017
Quantify the un-quantifiable

(c)
XAI: Explaining AI
Lecture Slides
http://concepts.gilb.com/dl958

A Serious 'Multi-dimensional Metrics Attack' on
Poor AI 'Academic and Standards' Thinking & Planning.
An analysis of published Principles for Managing and Standardizing AI, where about 10 AI Qualities like Safety and Transparency are shown to be quantifiable. This is prelude to rational thinking about the entire subject.
GilbFest Talk June 25 2019

(d) **IBM Cleanroom Method.**
MIlls and Quinnan Slides
http://concepts.gilb.com/dl896

Mills, H. 1980**. The management of software engineering: part 1: principles of software engineering.** IBM Systems Journal 19, issue 4 (Dec.):414-420.
Direct Copy
http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan

Includes Mills, O'Niell, Linger, Dyer,Quinnan p- 466

(e) **What is Wrong with Balanced Scorecard**, slides
http://concepts.gilb.com/dl135

(f)"**The Ohno Conspiracy" with**

Quality Function Deployment (QFD) detailed method analysis of method weaknesses.
http://concepts.gilb.com/dl954

(g) **'Design Sprint**
A Critical Analysis
and a Constructive Alternative'
3 Analytical Slides on 'Design Sprint'. 2 Critical Analysis and 1 with my alternative Startup Planning Week.
http://concepts.gilb.com/dl945

Appendix 24. Concept Glossary[24]

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.

Attribute: a characteristic of something. A quality, a cost, a function, anything which can describe and distinguish one artifact from another.

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.

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.

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.

---

[24] This Glossary should be consistent with any other Planguage Glossary. But in the interests of simplicity and freshness I have simply defined things in a simple sentence or so.

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 Constraint: A requirement specification, that demands or forbids something regarding a design.

Downstream, Upstream: downstream refers to a process to be carried out at a later stage. Upstream, a previous process.

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. + (added 270719)

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.

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.

Meter: a parameter which sketches major elements of a measurement process, for a particular Scalar Value or Cost.

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.

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.

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 Modeling of a system, and allows decomposition of problems so that critical 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.

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.

Target: a level of Value that we are aiming to reach. It includes Wish, Goal, Stretch.

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 benefit.

Appendix 99. Notes on editing the book and Versions.

25July 2021

Leanpub version

Fixed figure 1.3 caption

Changed references to Value series & SP from gilb.com to Leanpub

Started about 3 July 2019.

22July2019. First complete version ready. At Cabin

■ Last page.

EDIT 5AUG 2019

FAULT TO RECTIFY PAGE 195

Note 5 aug2019 I do not know where this figure is or what it is.

I PUT IN A CYCLE W IET ON 29 SEPT 2019

Figure: Architecture Engineering is a very systematic quantified discipline, which relies on quantification of Value Requirements (topic of this book) as an input.

August 14 2019

Moved TOC to front and put ship picture on cover, and edited risk defy x 2

240919: added 4 Gilb Summer books 2019 to references

Noted need edit. page 188 missing 2 figures (THE 2 FIGURES PUT BACK IN SEPT 29 2019) 203 the word constraints under figure needs edit, also page 208 missing fig., and see over page 195

September 29 2018 edit of missing illustrations

Putting in Ill and example numbers

To do

( ) Go to older copies and fix the 1.13? Tesla ill which is missing, get it from a backup copy

( ) Fig 1.14 ValPlan examples missing

Put in the word Chapter in every chapter, and numbered the sub chapters.

September 30 2019

1 Oct. 2019; full text edit of whole book. Clarifications and corrections .

I have not yet made this public either at twitter linkedin or gilb.com.

I have also received no feedback about the book. But I did send a version to selected GilbFest friends.

Full text edit whole book

1 oct lost diagram found around 15.2 use cases

———————————————

