A General Theory of Design: ‘Planguage’

By Tom Gilb, 2019   Version: 10 June 2019

This is a *claim* to have, and to share freely, a ‘**General Theory of Design**’. A ‘*Grand* Theory’ [1] possibly.

Part A. The Propositions.

Design Idea (‘A Design’): is a specification, made with the intent, to deliver some stakeholder values, using limited resources, within specified constraints.

Fundamental Design Ideas of Planguage. [16]

Proposition 1: Design Attempt

A ‘**design**’ (short form, noun, for ‘design idea’ a Planguage Concept)

attempts to *improve the distance*

towards a *required level of performance**,

from a known performance status level (benchmark),

to a required level of performance (target),

within resource constraints,

and *meeting* other specified constraints.

* **Performance**   **Concept *434 June 5, 2003**

System performance is an attribute set that describes measurably ‘how good’ the system is at delivering *effectiveness* to its stakeholders.

```
      Benchmark          Target
O———-<————————->——————->    Stakeholder Value X, Scale
          -The Design Gap-
```

Figure 1a. *The ‘design gap’ is an area of potential improvement, in the 'level of a stakeholder value’, for example reliability on an MTBF Scale. It is the core mission of a ‘design’ to try to fill this gap. A design is as good as the degree to which it promises, and then in reality, fills the gap, and thus reaches (or exceeds) the target level, the ‘success level’.*

The above illustration makes use of Planguage keyed icons described below, Part 7 and reference [7]

O———-**< [Design A]———->**——————-> Stakeholder Value X, Scale

-The Design Gap-
Remaining

***Figure 1b**. Design A is useful. It makes interesting, non-trivial progress towards our Target level. But, it is not* sufficient*. It can be (a.) re-specified to have greater impact, or (b.) replaced with another design, with sufficient impact, or (c.) we can add other designs - which increment the impact to the target, at least.*

Categories of Design Success.

Proposition 2. Usefulness.

A design is **useful** when it

in fact *does* improve the distance,

towards a required level of performance,

without *unacceptable* side-effects,

on other levels of performance,

and on budgeted resources.

O———-**<[Design A], Δ [Design B]>**_—————-> Stakeholder Value X, Scale

***Figure 2.** Design A, reached about 50% of the way to our Target level. It is useful, but not sufficient. The incremental addition (Δ) of Design B, allows us the reach the Target level, and beyond “_______>_” (satisficing). The underline (___) symbol expresses the visual degree of impact*

'Design A, Δ Design B’ is both useful, and sufficient.
*(Δ is a Planguage Icon for ‘Increment’ Concept *307)*

Sufficient means ‘reaching or exceeding the Target level, on time’.

More tersely, ***AΔB = >100%***

(Design A, with Increment Design B, gives more than 100% of the Target requirement level, so AΔB is sufficient design. AΔB satisfices.

We can also express these ideas numerically*:*
X <- A 50% *Design A impacts Scale X by 50% (from Benchmark, which is implied 0%)*

(X + ΔA) <- Δ B 60%   *Design B increments a further 60% (10% past the implied 100% level of the Target) on top of the increment we got from Design A (X ΔA).*

Proposition 3. Single Dimensional Success.

A design is **‘narrowly successful’** when it reasonably attains its expected (estimated) level of performance improvement, without us considering design side-effects (on other performance requirements, or budgeted resources)

```
O————<[Design A], Δ [Design B]>_—————>   Stakeholder Value X,  Scale
```

*Figure 3. The set of designs, **A +ΔB**, is ‘narrowly successful’ in single dimension. In this case we are not making a distinction between ‘the estimated impact’ indicates it might be successful; and ‘the actual current impact exceeds our targets’ This can be specified.*

Proposition 4. Multi-dimensional Success.

A design is **‘reasonably successful’** when it meets or exceeds its *expected performance improvement*, while having some useful side effects on other required performance requirement levels, and having expected-or-lower impact on budgeted resources.

```
—<——[A   ] [B   ]  O———<[Design A] +Δ[Design B]>_————>   Stakeholder Value X

                   O———<[A   ] [B   ]—————————>———————>  Value Y

                   O———<[A          ]—————————>———————>  Value Z

                   O———<—————————>———————>  Value R
```

Figure 4: Multi-dimensional success.

1. Design A & B only eat up about ⅔ of the only resource budget, at left
   —<——[A] [B] O
2. Design A&B have a nice (≈30%), which is a nice positive side effect.
   O———<**[A] [B]**—————————**>**———————> Value Y
3. A alone has a nice positive side effect on value Z
   O———<**[A]**—————————**>**———————> Value Z
4. And neither design A or B has any effect on Value R
   O———<—————————**>**———————> Value R
5. There are no negative side effects, at all, on specified Values.

The ‘O’ is the oval Planguage Keyed Icon for the ‘functionality’ (what system does) which has attached to it, the *value* and *cost* attributes.

Function **Concept *069 April 19, 2003 B**

A function is ‘what’ a system does.

A function is a binary concept, and is always expressed in action ('to do') terms (for example, 'to span a gap' and 'to manage a process').

Proposition 5. Comprehensive Success.

A design is **'very successful'** when it *exceeds* its expected performance improvement substantially,

and thus contributes even more (than expected), to meeting specified performance levels on time,

while also having some very useful side-effects, on other required performance requirement levels,

and also having lower-than-expected impacts on budgeted resources.

This is similar to the situation in Figure 4 above. With the exception of *substantially* exceeding the primary target level.

O— — -<- [Design A] __->___+Δ [Design B]____________— — — -> Stakeholder Value X

Figure 5: Substantially (_Substantially_— — ->) exceeding primary design target (__**->**__ attribute.

Proposition 6. Design to Attribute.

A design specification can be creatively, and intentionally, modified,

by a designer,

so that its resulting attributes (performance, resources, constraint satisfaction) are modified to be more successful

in satisfying the overall system requirements.

—<— — - —**[A'] [B']** **O**— — -**<[Design A'] +Δ [Design B']>**_— — — -> Value X

O— — -<**[A'] [B']**— — — -**>**— — — — — -> Value Y

O— — -<**[A']**— — — — — — — -**>**— — — — — -> Value Z

O— — -<— — — — — — — -**>**— — — — — -> Value R

Figure 6. In this visualization A’ and B’ are upgraded design specifications of Design A & B. The result has been estimated/measured to be:

a. Less-consuming of budgeted resources (about 50% versus ⅔ for A&B.,

b. Having greater impact on Value Y and Value Z, than A&B did

The main idea here is that, underperforming design specifications do not necessarily have to be replaced by different ones entirely, nor do they have to be ‘suffered’, as ‘all we can design’. Almost any initial design specification can be re-designed, yet seeing the initial basic concept in place, but tweaked with additions and changes so that it is both using less resources (‘design to cost), and delivering greater value (design to value). Or, more generally (values & resources) ‘design to **attributes**’.

The improved design specifications can be done, in early project stages, after initial estimations ([2] Impact Estimation method) have shown that the values and resource consumption are unacceptable; and before any serious costing and implementation of the initial design ideas. But, when we assume an agile incremental implementation of maybe 50 evolutionary value delivery steps (2% of budget step sizes), then the possibility of re-design at each and every step becomes an interesting option].

A well-documented and measured, 10 year industrial experience report of this, is found in the IBM Federal Systems Division ‘Cleanroom’ experiences [11].

Proposition 7. Design Satisfaction.

Design satisfaction occurs when,

first in theory,

later in practice,

a set of designs meets or exceeds performance design-targets,

within all constraints (resource budgets, and other specified constraints).

```
—<—[C ] [A’  ] [B' ] O——-<[Design A’     ] +Δ[Design B’     ]>_————->   Value X

                           O——-<[A’        ] [B’            ] [Δ C         ]>——————->  Value Y

                           O——-<[A’              ] [+Δ C——]>———————->  Value Z

                           O——-<[+Δ C             ]>———————->  Value R
```

Figure 7: By adding ‘Design C’, we managed to meet the required levels of Value Y, Z and R.

C was not necessary for meeting Value X, and the total resources of design A’, B’, and C did not exceed the budget level (left top, **—<—[C] [A’] [B'] O).** In fact some of the budgeted resource remains (**<**—).

Proposition 8. Design Survival.

Design 'Survival' occurs when:

first in-theory, *later* in-practice,

a set of specified designs, meets all *worst-acceptable-case* 'specified performance levels',

without exceeding any worst-acceptable-case *resource* budget limitations.

There is a critical difference between 'design **survival**', and 'design **success**'. 'W*orst-acceptable-case' is an informal reference to what we formally define as a 'Performance Constraint': the worst level of delivering a value that we have decided we will 'live with'.*

'Design Survival' means that our 'design set' meets all minimum conditions, that are 'needed to avoid failure' of the design set. This 'minimum' initially means the system is 'just barely alive', 'not declared dead': but by no means *thriving* and *successful*.

Value levels can get *better*, in this 'survival *range*', until we reach a well-defined 'Target level', defined as some kind of 'success'.

To enable us to determine if we have reached a 'survival' condition, **of a design set,** we need *first* to clearly define all 'constraints', and these need to be clear, unambiguous, testable, and, for *variables* (all values and resources), must be '**numeric**' [2, 3, 5]. Words like 'sufficient', 'good enough', 'satisfactory' have no useful precision, or agreed meaning.

This simple logical condition (well-defined constraints), for determining *viability of designs*, is too rarely met in my experience. This is, I believe, due to a lack of engineering discipline, taught academically - and enforced industrially.

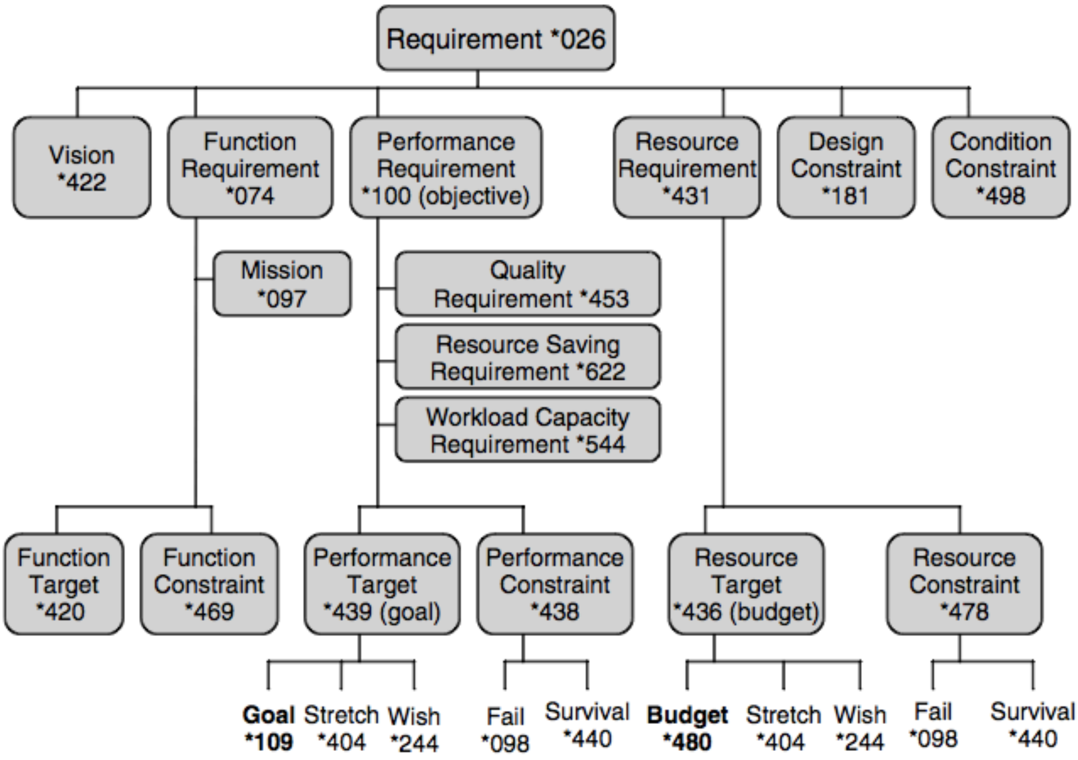

Requirement *026
Vision *422
Function Requirement *074
Performance Requirement *100 (objective)
Resource Requirement *431
Design Constraint *181
Condition Constraint *498
Mission *097
Quality Requirement *453
Resource Saving Requirement *622
Workload Capacity Requirement *544
Function Target *420
Function Constraint *469
Performance Target *439 (goal)
Performance Constraint *438
Resource Target *436 (budget)
Resource Constraint *478
Goal *109
Stretch *404
Wish *244
Fail *098
Survival *440
Budget *480
Stretch *404
Wish *244
Fail *098
Survival *440

Figure 8A. (Source Planguage Glossary, 'Requirement *026' in Competitive Engineering book [2].

There are a large number of constraint concepts in 8A, all defined in the Planguage Glossary [2].

Leaving aside, for the moment, *many* of the Planguage-defined constraint categories above (Figure 8A), I would like to initially introduce, a Scalar Constraint, for Performance Requirements.

This is the **'Performance Constraint'** (diagram above, definition below).

Performance Constraint **Concept *438 February 27, 2003, 2019**

A performance constraint specifies some upper and lower limits for an elementary scalar performance attribute.

These limits are either levels at which failure of some kind will be experienced, or levels at which the survival of the entire system is threatened.

Related Concepts [**Fail** *098]:

- Survival *440
- Catastrophe *602 '.'
- Range *552: See 'Failure Range'
- Must Do *539: Historical usage only
- Limit *606
- Level *337
- OK *669
- Tolerable *539

Figure 8B: a Planguage Concept Glossary list, of related Performance Constraint types. These exceed the 2 types in the diagram (8A) , 'Fail', and 'Survival'. There are in fact even more, and some clients create their own levels, to suit local culture (example Intel 'Landing Zone' for a Target area).

Keyed Icon [**Fail** *098]: **!**

"In context on scalar arrows: ---!--->O---!--->

A Failure *Range* would use multiple Fail icons: ----!!!!!!--->->

A failure range, just above a catastrophe range, is icon-ed like *this:*

O................!!!!!!!!!!!!!!!!![——————]>—————>
.Catastrophe. !Failure! [Tolerable] >Success <- Ranges

Figure 8C. Extract from Planguage Concepts Glossary, regarding the Keyed Icons for the Scalar Constraint 'Fail'. '…….' is the Catastrophe Keyed Icon ('.'), expressed as a range. The Success range beginning is denoted here with a single '>', but we can also express the ranges using a series of keyed icons

O................!!!!!!!!!!!!!!!!![-------------]>>>>>>>>>>
.Catastrophe. !Failure! [Tolerable] >Success

To the right of (!!!!!!!!!!!!!!!!!) the '**Failure**' level area is a '<u>survival</u>' condition, (or area, or range) known as '**Tolerable**'. This area continues until we reach the minimum Target (we are *aiming* for a Target level, but we must *avoid* 'non-survival' levels, and we 'just survive' in this unpleasant area known as 'Tolerable' [————-].

The '**>**' icon, just above the Tolerable level [——————], is the keyed Icon for the 'target' level called '**Goal**', which is popularly *defined* [12] as;

'we, the project,

can *commit* to delivering, *this* level of performance (or 'value'),

because all <u>necessary conditions are fulfilled</u>"

(Basically the conditions are design technical feasibility, economics, and compatibility with other commitments. See Planguage Concept Glossary 'Goal' for 8 types of structured conditions for being a Goal)

If is probably worth noting, that in Planguage, the exact specification of a Constraint is not merely dependent on the above ideas of a numeric level.

But, the more precise modelling of complex systems, requires additional specification detail, such as the following example:

<u>Security.Hack Identification:</u>
Scale % of hackers caught same day.

Tolerable 99%, [**Stakeholder** = EU Law, **Server Site** = Inside EU, **Data Type** = {Sensitive, Private}] by 8 June 2025

Goal 99.90%, [**Stakeholder** = EU Law, **Server Site** = Inside EU, **Data Typ**e = {Sensitive, Private}] by 8 June 2030

Figure 8D. In this constructed example, the minimum tolerable level (a scalar constraint that the designs must satisfy) of 99.00%, must be achieved by a deadline (June 8 2025), and must deal with, and *only* with, the specified set of conditions [**Stakeholder** = EU Law, **Server Site** = Inside EU, **Data Typ**e = {Sensitive, Private}].

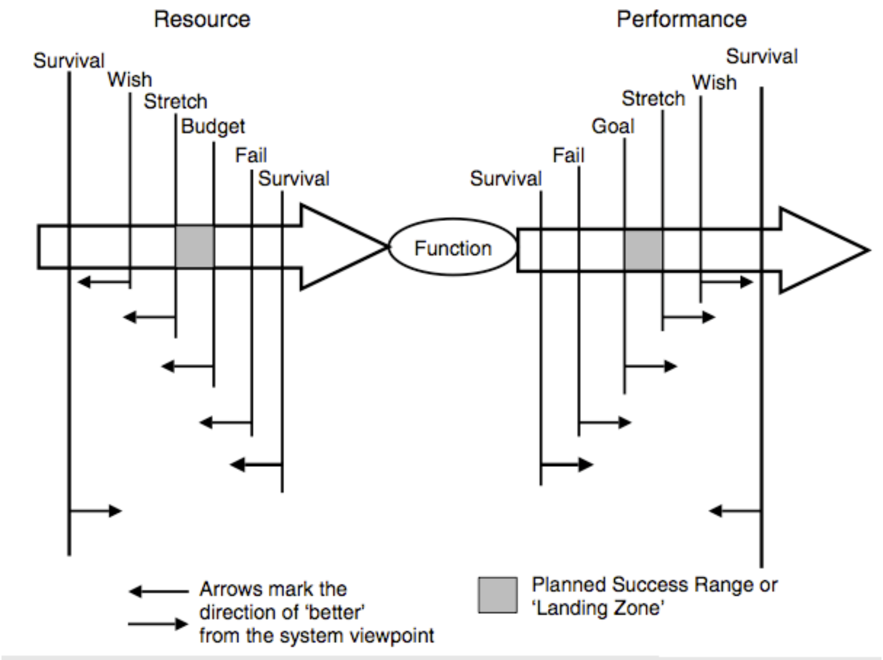
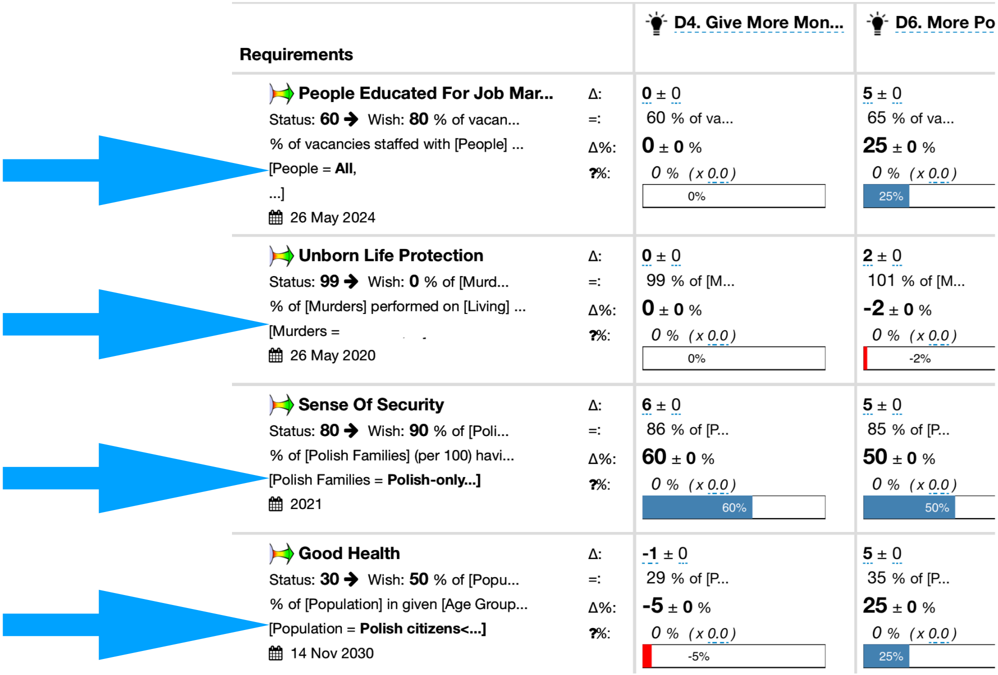

Resource
Performance
Survival
Wish
Stretch
Budget
Fail
Survival
Function
Survival
Fail
Goal
Stretch
Wish
Survival
Arrows mark the direction of 'better' from the system viewpoint
Planned Success Range or 'Landing Zone'

Figure 8 E: Drawn Plicons with a selection of constraint levels and target levels. Source CE [2]

Scale Parameters Allow Deeper Design Analysis

The reader might like to notice the detailed modelling power these scale parameters give, not only to the precision of a requirement, but as a consequence to *any statement about a design* which matches, and supports these parameters. We use this Planguage construction extensively in quantified evaluation of designs, using the Impact Estimation Table, which itself uses exactly such parameterized requirements to understand the efficiency of a corresponding design [2, 3, 5, 9].



Requirements		D4. Give More Mon...	D6. More Po
People Educated For Job Mar... Status: **60** ➔ Wish: **80** % of vacan... % of vacancies staffed with [People] ... [People = **All**, ...] 📅 26 May 2024	Δ: =: Δ%: ?%:	0 ± 0 60 % of va... **0** ± **0** % *0 % (x 0.0)* 0%	5 ± 0 65 % of va... **25** ± **0** % *0 % (x 0.0)* 25%
Unborn Life Protection Status: **99** ➔ Wish: **0** % of [Murd... % of [Murders] performed on [Living] ... [Murders = 📅 26 May 2020	Δ: =: Δ%: ?%:	0 ± 0 99 % of [M... **0** ± **0** % *0 % (x 0.0)* 0%	2 ± 0 101 % of [M... **-2** ± **0** % *0 % (x 0.0)* -2%
Sense Of Security Status: **80** ➔ Wish: **90** % of [Poli... % of [Polish Families] (per 100) havi... [Polish Families = **Polish-only...]** 📅 2021	Δ: =: Δ%: ?%:	6 ± 0 86 % of [P... **60** ± **0** % *0 % (x 0.0)* 60%	5 ± 0 85 % of [P... **50** ± **0** % *0 % (x 0.0)* 50%
Good Health Status: **30** ➔ Wish: **50** % of [Popu... % of [Population] in given [Age Group... [Population = **Polish citizens<...]** 📅 14 Nov 2030	Δ: =: Δ%: ?%:	-1 ± 0 29 % of [P... **-5** ± **0** % *0 % (x 0.0)* -5%	5 ± 0 35 % of [P... **25** ± **0** % *0 % (x 0.0)* 25%

Figure 8F A clip from an exercise in planning (Masterclass, Warsaw, May 2019). The designs (D4, D6) are estimated for potential effectiveness, against required values (the 4 'Requirements' on the left side, Good Health, etc.), which have Scale Parameters. Like '[Population]'. In the D6 design evaluation, of 25% effectiveness, this applies not to all Population elements, but specifically only to 'Population = **Polish Citizens**' (in Good Health).

The major *design theory* point, is that 'evaluation of design effectiveness' can be made *very selectively*, with respect to a selected set of dimensions, from all dimensions in the total system model.

This 'Selective Dimension Design Evaluation' has several interesting characteristics: Agile Design and Lean Dynamic Design.

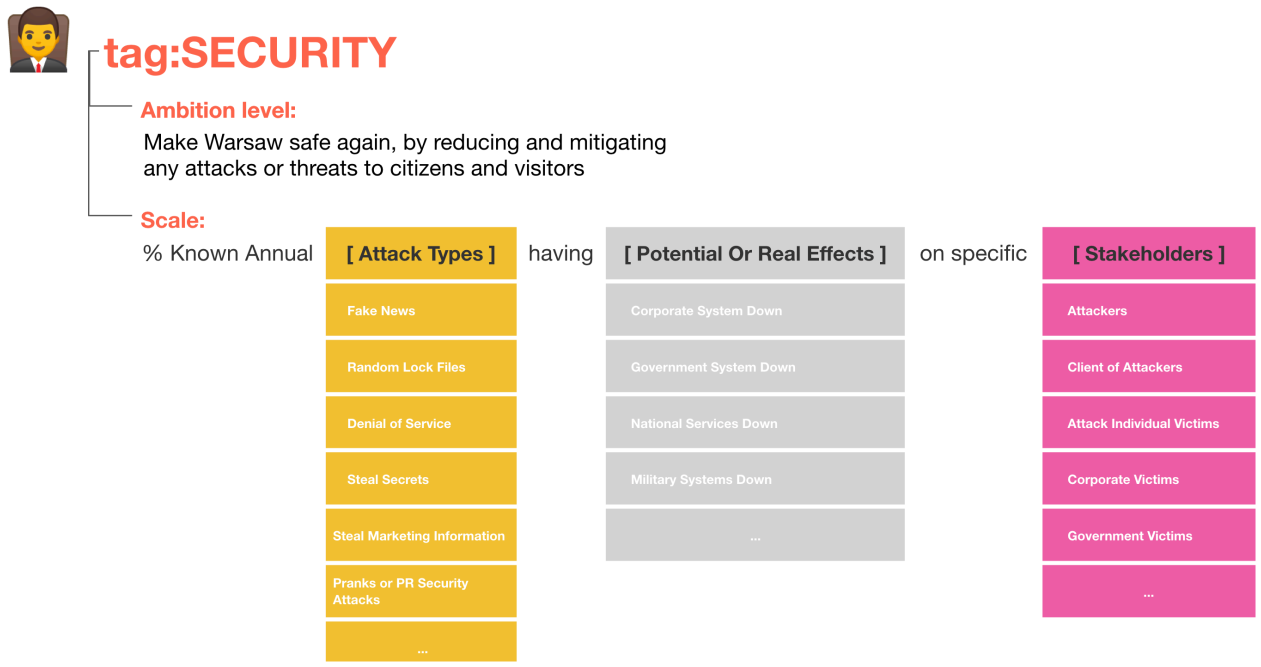
1. We can evaluate 'very critical sets' of parameters *first* (before all other valid combinations)
2. We can evaluate *which* designs, and sub-designs are most *effective*, for our *most-critical* requirements. Selective Designs for Critical Requirements.
3. We can *then* **deploy** the most cost-effective designs early, in an incremental sequence
4. This will give best value for 'resources used to date'.
5. It will build *confidence* in successful designs, in preparation for *scaling up,* or for *spreading* to other Scale Parameter dimensions (like other cities, other types of people)
6. It will maximize the profitability of the project (best and cheapest designs deployed early, and resource cutoff is possible, when we reach 'diminishing returns on investment'.

Big Bang design-thinking goodbye. Welcome 'Dynamic Design'. Or, as I also like to put it 'Agile' as it should be' (*with* clear requirements and design) and as Agile was always meant to be by me [9, the cited (by several Manifesto signers) foundation for the Agile Manifesto [14]: but they forgot some stuff, like 'design'].

Three-Dimensional Design Space

Here is a visualization of the design space, into which we can attempt to find suitable designs for various requirement dimensions. Let me call this '3D-Design Thinking'.

1. The Value or resource attribute dimension (usually about 15 of these at top level)
2. The [Scale Parameter] Level dimension. Usually 3 to 7 of there per Scale.
3. The Scale Parameter Attribute level dimension. About 5 to 20 of these usually.



tag:SECURITY

Ambition level:

Make Warsaw safe again, by reducing and mitigating any attacks or threats to citizens and visitors

Scale:

% Known Annual [Attack Types] having [Potential Or Real Effects] on specific [Stakeholders]

[Attack Types]	[Potential Or Real Effects]	[Stakeholders]
Fake News	Corporate System Down	Attackers
Random Lock Files	Government System Down	Client of Attackers
Denial of Service	National Services Down	Attack Individual Victims
Steal Secrets	Military Systems Down	Corporate Victims
Steal Marketing Information	...	Government Victims
Pranks or PR Security Attacks		...
...		

Figure 8G. An illustration of the [Scale Parameter] dimensions. There are 144 theoretical combinations of single parameters, and we can do 2 or more at a time. Illustration designed by anna@karlowska.pl 2019-05-28.

Proposition 9. ‘Design *sensitivity*’ to requirement modification.

The moment any **requirement** specification is **changed*****,

in level, timing, or constraint

there is a *risk***** that *any and all* design specifications made successfully*, *before* that,

are, wholly or partially, **invalidated****.

* successfully: the design spec gives a ‘useful’ impact and cost.

** **invalidated**: might be useless design, and would need improved design-specification, to be useful again.

***For example, a change of ‘[Scale-parameter] attribute choice’, see below Fig 9 A&B.

**** (a risk) not a *certainty* of invalidation. The result might *even* be ‘better’.

Design Specification Impacts are a function of…

Both the *estimate* of design-attribute impacts, and the *actual delivered impacts* are a function of:

1. The exact and detailed *requirement specification*, (values, and resource budgets).

2. The exact design specification *interpretation**
   1. first ‘in theory’, for estimation of design attributes,
   2. then ‘in practice’, the interpretation of the design specification itself, for the process of implementation.

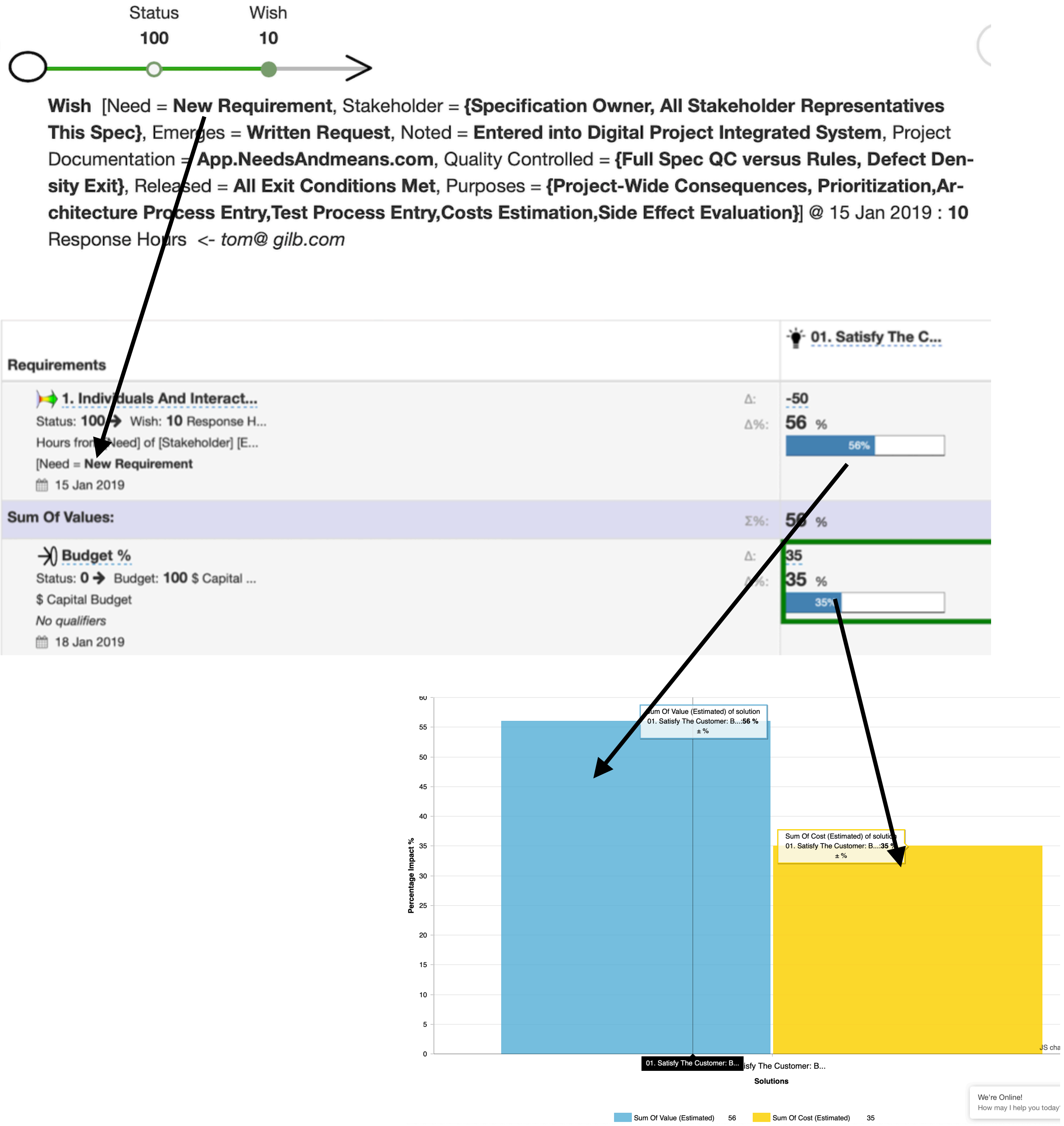
Design Spec Supply Chain Corruption: The ‘Telephone Game’ Effect (Chinese Whispers, UK) On Design.

* Different people, might interpret an ambiguous design specification, in different ways. This of course leads to a different set of impacts of the design they implement. This effect is not predictable, since even the most perfectly unambiguous design specification, can be misinterpreted by accident, or wilfully, by one-or-more human intermediaries, between the design spec and real implementation.

Think of the ‘Butterfly Effect’ analogy.

“In chaos theory, the **butterfly effect** is the sensitive dependence, on initial conditions, in which a small change, in one state of a deterministic nonlinear system, can result in large differences, in a later state.” Wikipedia.

Consequently any change, in the requirements & design specification, risks changing the estimates, and later, can change the real outcome of values and costs.

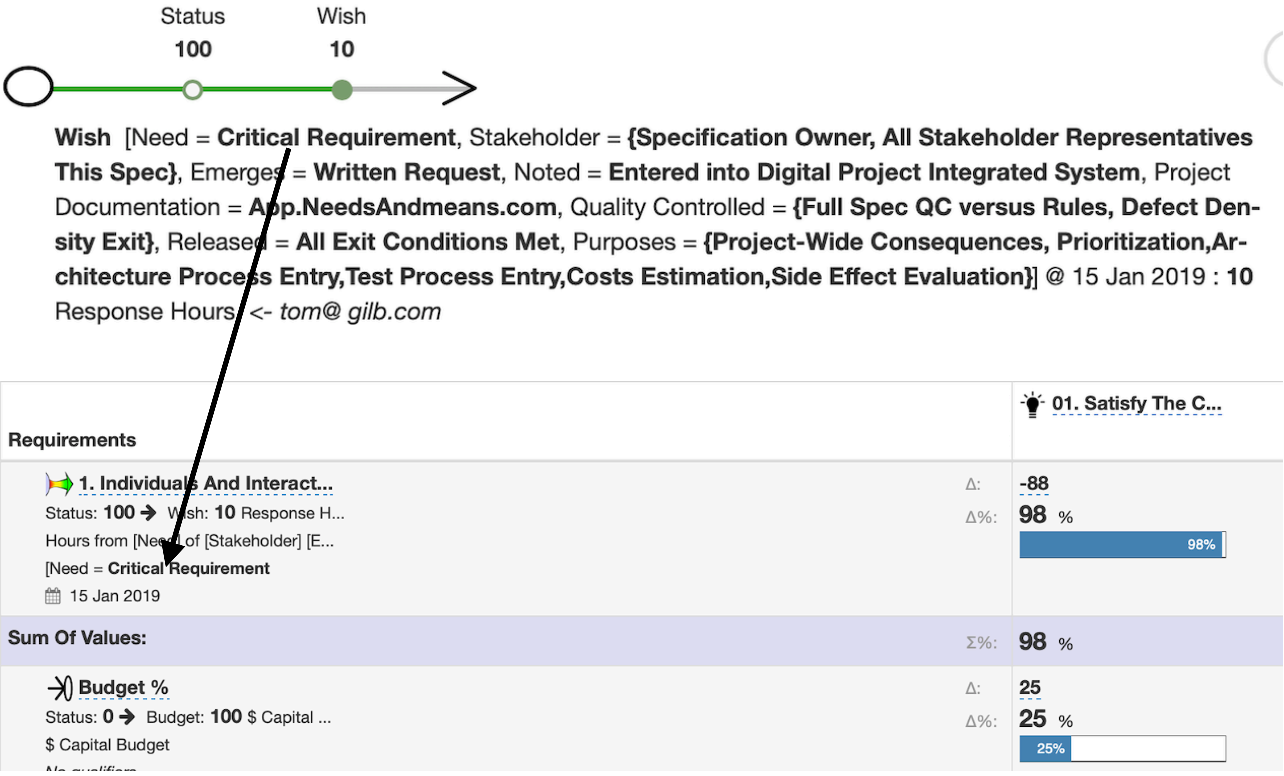


Wish [Need = **New Requirement**, Stakeholder = **{Specification Owner, All Stakeholder Representatives This Spec}**, Emerges = **Written Request**, Noted = **Entered into Digital Project Integrated System**, Project Documentation = **App.NeedsAndmeans.com**, Quality Controlled = **{Full Spec QC versus Rules, Defect Density Exit}**, Released = **All Exit Conditions Met**, Purposes = **{Project-Wide Consequences, Prioritization,Architecture Process Entry,Test Process Entry,Costs Estimation,Side Effect Evaluation}**] @ 15 Jan 2019 : **10** Response Hours <- *tom@ gilb.com*

Requirements		**01. Satisfy The C...**
1. Individuals And Interact... Status: **100** → Wish: **10** Response H... Hours from [Need] of [Stakeholder] [E... [Need = **New Requirement** 15 Jan 2019	Δ: Δ%:	**-50** **56** % 56%
Sum Of Values:	Σ%:	**56** %
Budget % Status: **0** → Budget: **100** $ Capital ... $ Capital Budget *No qualifiers* 18 Jan 2019	Δ: Δ%:	**35** **35** % 35%

60
55
50
45
40
35
30
25
20
15
10
5
0
Percentage Impact %
Sum Of Value (Estimated) of solution
01. Satisfy The Customer: B...:56 %
± %
Sum Of Cost (Estimated) of solution
01. Satisfy The Customer: B...:35 %
± %
JS cha
01. Satisfy The Customer: B...
isfy The Customer: B...
Solutions
We're Online!
How may I help you today
Sum Of Value (Estimated) 56
Sum Of Cost (Estimated) 35

Figure 9A: With the Requirement (**1. Individuals and Interaction**) '**Need**' scale parameter set to '**New Requirement**' we estimated the impact of the design '**01. Satisfy the Customer**' [14], to satisfy the Value (moving from Status 100 down to 10 scale units) as '56%' of the way to the Wish level. And we estimated the cost on the '**Budget**' resource specification, as '35%' of the Budget.



Wish [Need = **Critical Requirement**, Stakeholder = **{Specification Owner, All Stakeholder Representatives This Spec}**, Emerges = **Written Request**, Noted = **Entered into Digital Project Integrated System**, Project Documentation = **App.NeedsAndmeans.com**, Quality Controlled = **{Full Spec QC versus Rules, Defect Density Exit}**, Released = **All Exit Conditions Met**, Purposes = **{Project-Wide Consequences, Prioritization,Architecture Process Entry,Test Process Entry,Costs Estimation,Side Effect Evaluation}**] @ 15 Jan 2019 : **10** Response Hours. <- *tom@ gilb.com*

Requirements		**01. Satisfy The C...**
1. Individuals And Interact... Status: **100** ➔ Wish: **10** Response H... Hours from [Need] of [Stakeholder] [E... [Need = **Critical Requirement** 15 Jan 2019	Δ: Δ%:	**-88** **98** % 98%
Sum Of Values:	Σ%:	**98** %
Budget % Status: **0** ➔ Budget: **100** $ Capital ... $ Capital Budget	Δ: Δ%:	**25** **25** % 25%

Figure 9b: With the Requirement (**1. Individuals and Interaction**) '**Need**' scale parameter set to '**Critical Requirement**' we estimated the impact of the design '**01. Satisfy the Customer**' [14], to satisfy the Value (moving from Status 100 down to 10 scale units) as '98%' of the way to the Wish level. And we estimated the cost on the '**Budget**' resource specification, as '25%' of the Budget.

The point with this simple, and 'constructed', example, is that **even the smallest change in the set of requirements and design** (in this case in requirement scale-parameter attribute ('Need =___') **can dramatically affect the estimation (and similarly also the real measured effects) of the design impacts.**

This 'design sensitivity to spec changes' is one of many good reasons to plan *digitally* [8], at least with a spreadsheet, so that the designer can 'see' the *overall effects*, of a few subtle design changes, of estimation changes, and of value-delivery-cycle ('sprint') measured feedback.

From this point of view, the 'yellow stickies' are not serious planning tools for complex systems. Not even when they ares digitized in an app :) !

Proposition 10. Scientific Experimental Evaluation of Multiple Designs

Designs are best *evaluated* (estimations); **sequentially, and incrementally**

and then also *validated* (measurements), **sequentially, and incrementally**,

so that we can better understand the 'design cause' of the system attribute effects.

Most systems will be made, and kept alive, with a set of discrete and independent designs.

The number of individually implementable designs can be large (100, or many more).

In practice, designers know very little, about most of the many value & cost attributes, of even one *single* design.

In general, the 'design attribute knowledge' is simply not collected, recorded, available, and useful.

In addition, there is the 'problem of understanding the process of design cumulation'. Well understood to the engineering culture in general.

Some Principles Design Additivity.

1. We do not know exactly the attribute states of the system, which we are going to add our single design into.
2. We do not know exactly (or even approximately, even order-of-magnitude) what will be the additive effect of incrementing a next design to an unknown set of previously-implemented designs. It can be useful, to try to estimate, anyway, but there is no certainty; only hope.
3. We can *measure* the state-of-the-attributes of the incremented system, *before* we implement our 'next design' increment. Measurement is never *certain*, but it beats estimation.
4. But there is no guarantee that this set of cumulated system attributes, will be a *stable* set of attributes, since they can be impacted greatly by external factors, over which we have no control, and even less predictive knowledge.
5. We can simply add a design increment, and see what happens. Then we can measure the resulting attributes, and possibly observe if they are stable. And observe (measure) if they change, when selected external and internal variables are changed.
6. But we have no guarantees, that a subsequent design addition will not do unpredictable and negative damage to any 'hard won' attribute status, observed beforehand.

These principles occur to me as general, obvious, observable, and irrefutable, with few, uninteresting exceptions. I just brainstormed them 2019-6-9. TsG

Scientific Design Experiment

Now this the same situation both all engineers and all scientists find. And that is why they know they must make use the scientific experimentation principle:

1. Do 'design implementation' one step at a time.
2. Keep all other factors constant.
3.Then, *hope* that the effects you observe, from the design increment, were *really* caused by *that* design.

A great practical engineering example of this, is the Cleanroom method [11].

Quinnan describes the process-control-loop used by IBM FSD to ensure that cost targets are met.

'Cost management. . . yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance.

Design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method consists of developing a design, estimating its cost, and ensuring that the design is cost-effective.' (p. 473)

He goes on to describe a design iteration process trying to meet cost targets by either redesign or by sacrificing 'planned capability.'
When a satisfactory design at cost target is achieved for a single increment, the 'development of each increment can proceed concurrently with the program design of the others.'

'Design is an iterative process in which each design level is a refinement of the previous level.' (p. 474) [11] IBM SJ 4/80

From the Cleanroom Method Father, Harlan Mills [11], we know that this above design process was key to achieving years, of large complex military and space software projects, being on-time and under-budget. So this is a design process well worth noting. It is also 'agile, the way it should be', as I put it.

Robert Quinnan, IBM FSD

Proposition 11. There is a logical sequence, often iterative, of analytical design-related processes, which help us find good enough designs.

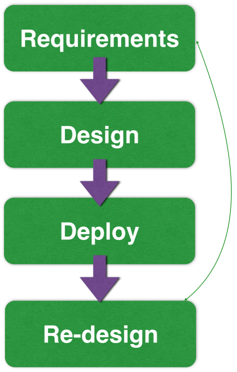


Figure 11A. The basic design iteration. Each step is a learning step.It can learn some things about the design environment, and use that learning to improve the design.

Here is a more-detailed sequence of design related processes. [15C]

1. Environment Scope helps identify stakeholders.
2. Stakeholders have values and priorities
3. Values have many dimensions
4. Stakeholders determine value levels
5. Design hypotheses should be powerful and efficient ideas, for satisfying stakeholder needs
6. Design hypotheses can be evaluated quantitatively, with respect to all quantified objectives and resources
7. Designs can be decomposed, to find more efficient design subsets, that can be implemented early
8. Designs can be implemented sequentially, and their value-delivery, and resource costs, measured
9. Designs that unexpectedly threaten achievement of objectives, or excessive use of resources, can be removed or modified.
10. Designs that have the best set of effects on objectives, for the least consumption of limited resources, should generally be selected for early implementation.
11. A design increment can have unacceptable results, in combination with previous increments, and they, or it, might need removal or modification
12. When all objectives are reached, the process of design is complete: except for possible optimization of operational resources, by even-better design.
13. When deadlined and budgeted implementation-resources are used up, it might be reasonable to negotiate additional resources; especially if the incremental values are worth the additional resources.
14. When deadlined and budgeted implementation-resources are used up, it might be reasonable to negotiate additional resources; especially if the incremental values are worth the additional resources.

The paper [15C] gives more detail for each of the 14 design-related processes.

A take-away from this set of design-related processes, is that there is *nothing simple*, like

Requirements —> Design

Proposition 12. The priority of alternative designs can be determined by a variety of prioritization policies; most of which are based on objective ‘values for resources’, with regard to risk. [16]

This subject is complex. But I will summarize it here. The detail, in 60 pages, is in [16C], Value Planning book chapter on Prioritization.

Principles of Design Prioritization.

1. **Design Priority**, ‘the *selection* of the next design to be incremented into the system’, depends on
   a. The subjective chosen viewpoint (Prioritization Policy) of the stakeholders empowered to prioritize, and
   b. The richness and quality of information about the design, and the corresponding requirements.

2. The **sequential ‘next design’ choice** can be computed, at each step, based on the following digital information:

a. The remaining gap in values, to scalar constraints (like ‘Tolerable Levels),
b. Then when all Scalar constraints are met, the gap to Targets can be applied.
c. The remaining resources, of various types, to Budgeted level

These considerations will alert us to the needs un-met, and resources available. Opportunities and necessities.

3. Then we can **look for available design candidates** and consider the following factors:

d. The estimated value delivery, to each residual value requirement gap
e. The set of resource costs necessary to deliver that design, compared to remaining resources.
f. The set of values-to-resources ratio: relative ‘efficiency’ or ‘profitability’ of the choice.
g. The worst-case uncertainty: the lowest value levels, the highest cost levels.
h. The credibility level (0.0 to 1.0) based on estimation evidence and source quality.

3. An important idea, different from conventional thinking about priorities [16A], is that *design priorities are not at all fixed or static*. They are highly dynamic, subject to re-determination in real time, based on the many factors above. And the changing nature of the many factors.

A useful analogy is both chess, and our body.

The Chess player must constantly re-prioritize based on the clock, the last move of the opponent, and the rules of the particular type of chess. They can rarely control things several moves ahead.

Our body is constantly computing dynamic priority for survival, then for success factors, and decides when and how much to apply designs of food, sleep, liquid, air, sex, temperature; to help us survive and to thrive.

Design prioritization is essentially the same nature, if it is advanced.

All this is already automated in the planning tool ValPlan.net (since about 2015).

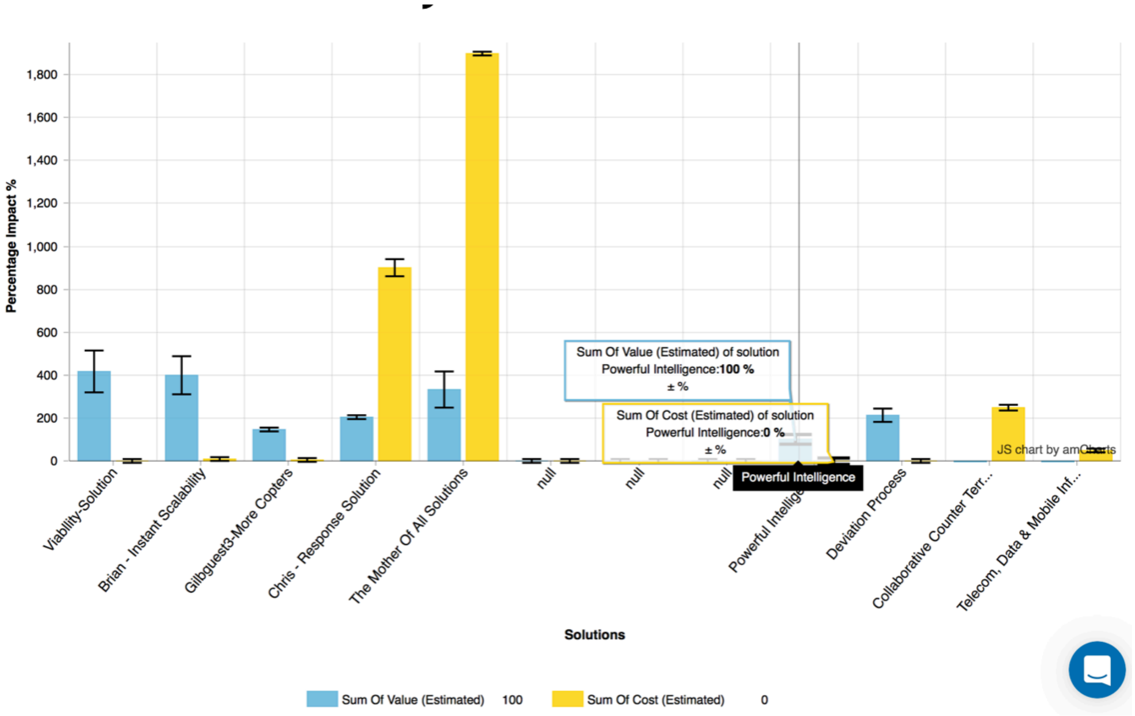
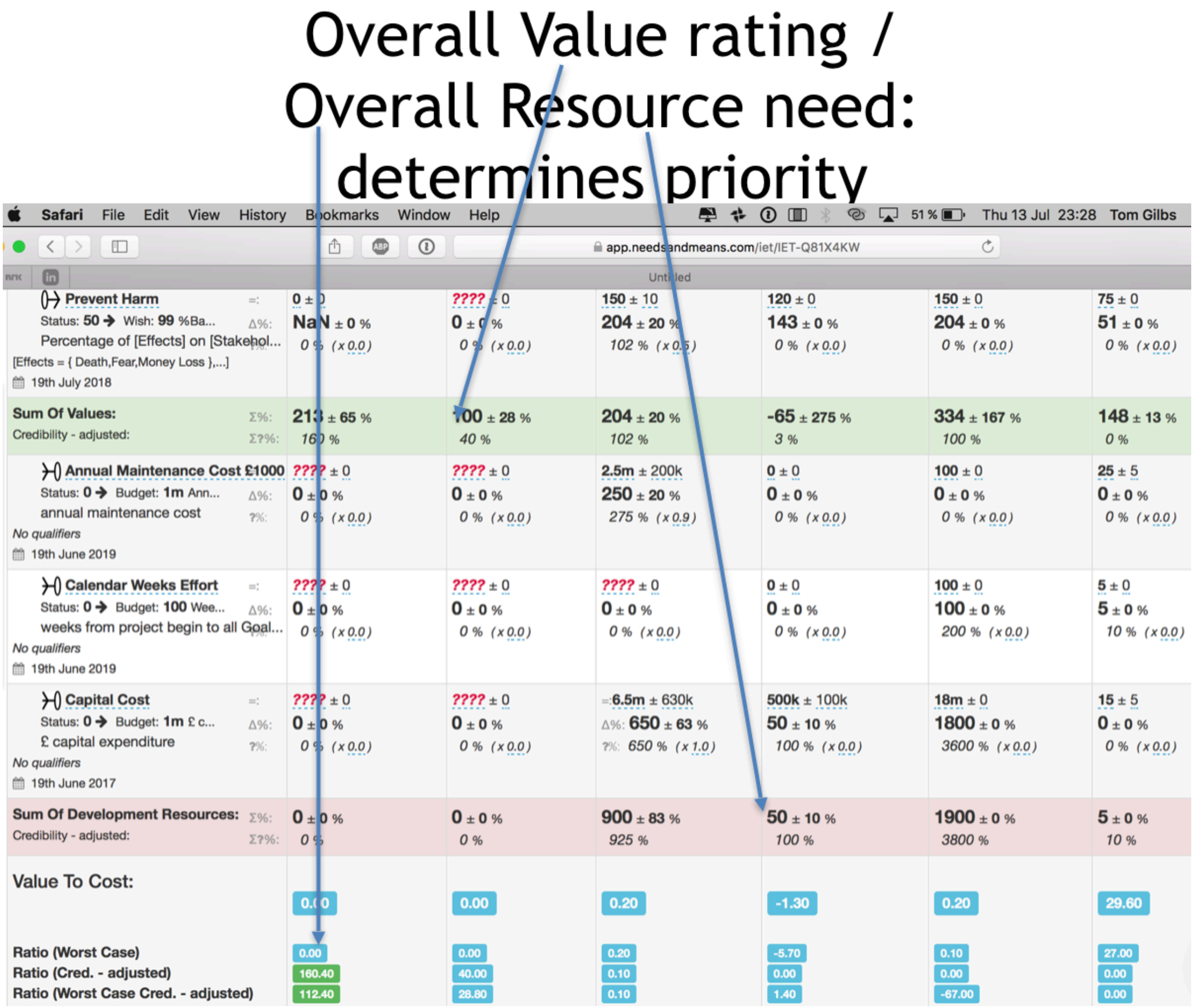


Figure 12A. The various design options are estimated (about 10 values, and 5 resource constraints) on an Impact Estimation Table. Together with information on uncertainty and credibility. They are then sorted left to right by values/resources with respect to risks (uncertainty, credibility). Leading to a strong suggestion that the design tagged 'Viability-Solution' should be implemented next. It looks like 'it won by a hair', based on the worst case value level being a little bit higher (the I bar lower part).



Overall Value rating / Overall Resource need: determines priority

⟩-⟩ **Prevent Harm** Status: **50** → Wish: **99** %Ba... Percentage of [Effects] on [Stakehol... [Effects = { Death,Fear,Money Loss },...] 19th July 2018	=: Δ%: ?%:	0 ± 0 **NaN** ± 0 % *0 % (x 0.0)*	???? ± 0 **0** ± 0 % *0 % (x 0.0)*	150 ± 10 **204** ± 20 % *102 % (x 0.5)*	120 ± 0 **143** ± 0 % *0 % (x 0.0)*	150 ± 0 **204** ± 0 % *0 % (x 0.0)*	75 ± 0 **51** ± 0 % *0 % (x 0.0)*
Sum Of Values: Credibility - adjusted:	Σ%: Σ?%:	**213** ± 65 % *160 %*	**100** ± 28 % *40 %*	**204** ± 20 % *102 %*	**-65** ± 275 % *3 %*	**334** ± 167 % *100 %*	**148** ± 13 % *0 %*
⟩-⟨ **Annual Maintenance Cost £1000** Status: **0** → Budget: **1m** Ann... annual maintenance cost *No qualifiers* 19th June 2019	Δ%: ?%:	???? ± 0 **0** ± 0 % *0 % (x 0.0)*	???? ± 0 **0** ± 0 % *0 % (x 0.0)*	2.5m ± 200k **250** ± 20 % *275 % (x 0.9)*	0 ± 0 **0** ± 0 % *0 % (x 0.0)*	100 ± 0 **0** ± 0 % *0 % (x 0.0)*	25 ± 5 **0** ± 0 % *0 % (x 0.0)*
⟩-⟨ **Calendar Weeks Effort** Status: **0** → Budget: **100** Wee... weeks from project begin to all Goal... *No qualifiers* 19th June 2019	=: Δ%: ?%:	???? ± 0 **0** ± 0 % *0 % (x 0.0)*	???? ± 0 **0** ± 0 % *0 % (x 0.0)*	???? ± 0 **0** ± 0 % *0 % (x 0.0)*	0 ± 0 **0** ± 0 % *0 % (x 0.0)*	100 ± 0 **100** ± 0 % *200 % (x 0.0)*	5 ± 0 **5** ± 0 % *10 % (x 0.0)*
⟩-⟨ **Capital Cost** Status: **0** → Budget: **1m** £ c... £ capital expenditure *No qualifiers* 19th June 2017	=: Δ%: ?%:	???? ± 0 **0** ± 0 % *0 % (x 0.0)*	???? ± 0 **0** ± 0 % *0 % (x 0.0)*	=:6.5m ± 630k Δ%: **650** ± 63 % ?%: *650 % (x 1.0)*	500k ± 100k **50** ± 10 % *100 % (x 0.0)*	18m ± 0 **1800** ± 0 % *3600 % (x 0.0)*	15 ± 5 **0** ± 0 % *0 % (x 0.0)*
Sum Of Development Resources: Credibility - adjusted:	Σ%: Σ?%:	**0** ± 0 % *0 %*	**0** ± 0 % *0 %*	**900** ± 83 % *925 %*	**50** ± 10 % *100 %*	**1900** ± 0 % *3800 %*	**5** ± 0 % *10 %*
Value To Cost:		0.00	0.00	0.20	-1.30	0.20	29.60
Ratio (Worst Case)		0.00	0.00	0.20	-5.70	0.10	27.00
Ratio (Cred. - adjusted)		160.40	40.00	0.10	0.00	0.00	0.00
Ratio (Worst Case Cred. - adjusted)		112.40	28.80	0.10	1.40	-67.00	0.00

Figure 12B. The lower half of an Impact Estimation Table, showing all the prioritization data-source elements. The Uncertainty and credibility (±26%) numbers (x 0.9). The ???? Unknowns. And at the very bottom, the 4 types of values/costs ratios. Hidden from this view are most of the about 10 value estimations, but they are summarized in the 'Sum of Values' line. Detail on this Impact Estimation table method, is found in the Competitive engineering book [2].

Part B. The Planguage Concept Glossary [12]

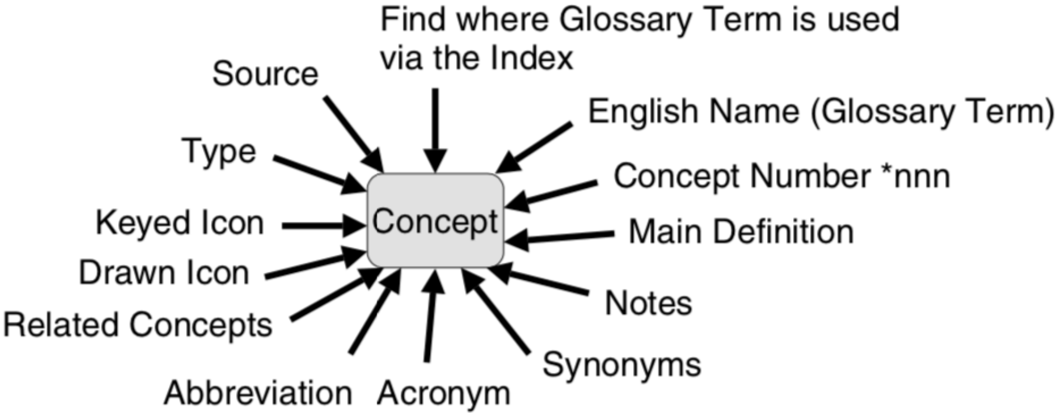


Figure B1. The many pointers to a concept.

The Planguage Concept Glossary is the conceptual foundation for my design theory. It pins down about 800 related concepts, in an integrated way. That is, every concept is *completely aware* of all the others. In fact defined concepts are used to define other concepts, more accurately.

I have always consulted dictionaries, and professional standards before defining my concepts. But they are not of much help in the precision needed, and the completeness needed, for this task. The standards reek of committee compromise, and the dictionary is of ancient culture.

I do have a few decades of experience, telling me that Planguage is defined by a stable set of concepts, and useful ones. And most of them I feel are better than what most people re applying.

One major innovation has been my design of the 'Concept definition', rather than a word or term definition. Arguing about the 'meaning of a word' is a wasteful pursuit. It is the core concepts that are important to a design theory. So a wide variety of useful pointers can be used to access the concepts, even numbers, icons, and non-english words, even British ones :) And synonyms too.

The word 'design' itself posed challenges, since it is used both as an adjective, 'to design' and a noun ('a design'). So I settled on 'design idea' for the noun, as my textbook pointer to the concept.. But I acknowledge that we use many synonyms for 'design idea', such as architecture, solution, strategy, tactic, idea, and 'means objective'.

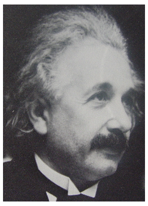
So I find it very helpful to be clear that, whatever we call a 'design idea' it is a perceived 'means to an end'. Sometimes it is a great 'teaching relief' to simply express it graphically: like….

```
        [THE MEANS TOWARD THE END]
O——<————————————————>——————————>  Value X
                    End
```

Figure B2. The ' [THE MEANS TOWARD THE END]' is the Keyed rectangle symbol for a 'Design Idea' Which graphically delivers enough effect to Value X Requirement, to reach the Goal (->-) level.

I was quite surprised when, awakened by Ralph Keeney (Value Focused Thinking), I realised that **one person's design, *becomes* another person's requirement.**

And that this 'dual personality' of a design can persist a number of levels in the design process chain.



No wonder, Einstein observed: **"Perfection of means, and confusion of ends, seems to characterize our age"**

Part C. The Planning Language, 'Planguage' [2, 17]

'Planguage: an 'engineering' language and process for real software and systems engineering - not 'programming' [17A]

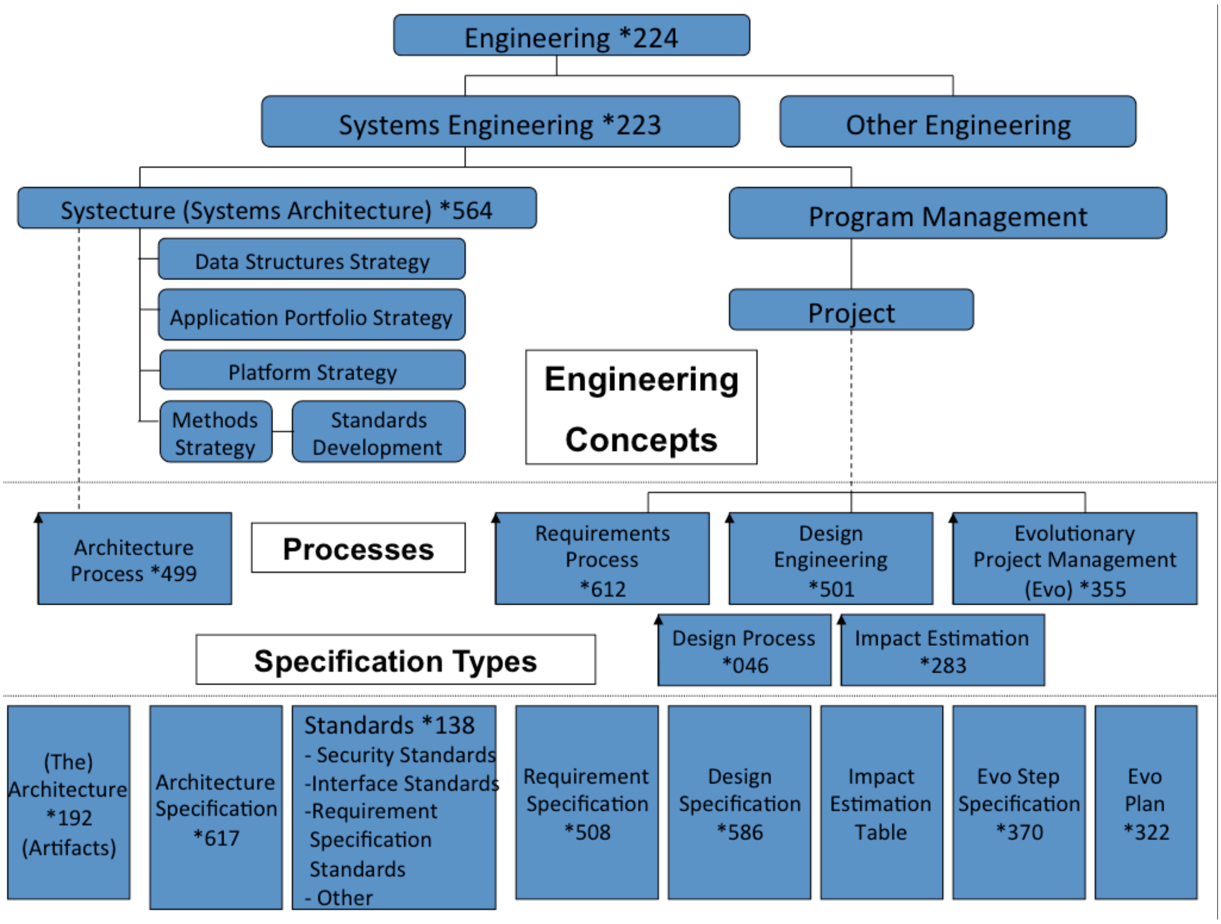


Figure C: An overview of artifacts in Planguage. sSource Competitive Engineering [2]

'Planguage' is a general-purpose, systems engineering, planning language; for any system, including software systems. Planguage scope is requirements, design, project management, and quality control.

It has been developed and practiced for decades (since 1960s). It is open source; anybody can use it for anything, in whole or part, freely. It is a large integrated 'toolbox', containing hundreds of distinct tools. Any set of these tools can be added to any other set of tools, or any framework. In particular, it is suitable as a set of 'practices' to evolve one's own method.

Planguage was designed to be interpreted by computers. The earliest automation was done by Prof. Lech Krzanik in 1978-9, on an Apple II in Forth, and published in his PhD Thesis, as well as in 'Principles of Software Engineering' (1988, 'Aspect Engine'). Many automated tools have been made since, by Kai Gilb, and our clients, to support its use. The latest is www.ValPlan.net
Planguage was also designed to be 'translated' easily into any nation's language. It includes a *graphical* representation language, as part of this [17B].

The *central* distinguishing characteristic of Planguage is it's **ability to directly integrate any quality** (any '-ility', not just reliability) statement quantitatively **into the requirements, the designs, the project management, and the quality-control methods** it contains.

The *second* distinguishing characteristic of Planguage is that it allows and encourages very **'rich' planning specification of the background information for each individual requirement and design**.

This supports risk management, change management and dynamic prioritization.

A *third* distinguishing characteristic is a systematic **devotion to *clarity* and *intelligibility* of specification.**

Ambiguity, and lack of testable clarity is unacceptable. Even for 'soft' characteristics. Metrics, measurability and frequent numeric feedback about performance and costs is a primary notion.

Planguage, with its project management component. 'Evo' (Evolutionary Value Delivery) is recognized as the 'grandfather' of Agile methods. Both in terms of earliest publication (1970s, 1980s [14, see references there], and by Agile method developers (Sutherland, Beck, Cohn and many others]. The Agile method developers mainly refer to Principles of Software Engineering Management, 1988 [9B in particular]

The two largest-scale adoptions of Planguage were at HP (Corporate Wide from 1988) [19], and is at Intel (over 21,000 engineers, over 10 years) [20]. A body of literature exists for this, including academic studies.

Other noteworthy adoptions documentable, but often less than Corporate-wide, sometime lasting only a few years include IBM (Corporate Quality Policy, CMM 4), ICL (1982, top management, sw product development), Boeing (1990, aircraft engineering QC, Process Error Prevention method), McDonnell Douglas (aircraft engineering, 1998-90), Citigroup (2003-2006), Credit Suisse, JP Morgan, Union Bank of Switzerland, Philips Medical Systems, Ericsson (ERA, 1990s), Nokia & Symbian, Microsoft (Test). There are many smaller and lesser known organizations and single projects for which we have published case studies.

Good sources of detailed Planguage examples, case studies and references to original material is in the 'Value Planning' book [5]

Part D. The Principles [2], [18]

Principles are short statements, which summarize wisdom. Principles can give strong guidance towards success and for avoiding failure. Good principles will be useful for a broad range of problems. Principles are sometimes called Engineering Heuristics [21, Koen]

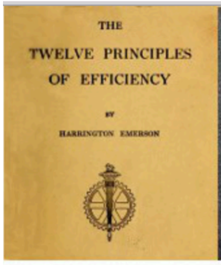
My own criteria for a principle is that, in addition to being ‘powerful’, they are eternally true. They have a very long ‘half life’. They are independent of current technology. I ask myself, would this principle have been valid in Egyptian Pyramid times, and Roman Aqueduct times ? Has this principle worked for me since 1958- to date? Do I see any reason why the principle would become invalid in 1,000 years time? If OK, then I would use them.

When I generate a principle, they are just a way of summarizing my own insights, experience and common sense. Of summarizing what I need to teach others.

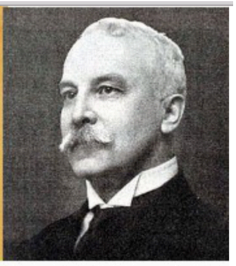
But the rate at which I can do this ‘generation of principles’, usually surprises me. Who cares, as long as they work, and survive.

In this case my principles help to define, and support, my design theory.

- “As to methods, there may be a million and then some, but principles are few.
- The man who grasps principles can successfully select his own methods”.
- **-Harrington Emerson**



Book Cover



Harrington Emerson

August 2, 1853 September 2, 1931) was an American efficiency engineer and business theorist,[1] who founded the management consultancy firm Emerson Institute in New York City in 1900.

Reference [15A] ‘Ten Design Principles’, 2006, is perhaps a good summary of design principles for the purposes of this paper.

- 2. **The ‘Valid Design’ Principle**
  - **A *valid* design must contribute to performance goals, within all constraints.**
  -
  - *This has the following implications:*
  - *you must be able to prove that a design does not violate any defined constraints;*
  - *you must be able to prove that a design contributes, to at least one yet unfulfilled performance requirement;*
  - *the design cannot be justified, if its only contribution has been made by another accepted design already;*
  - *a change in the numeric performance level, required for a performance requirement, can invalidate a previously acceptable design, or make a previously discarded design valid.*

Here is a random selection of the Ten Design Principles.

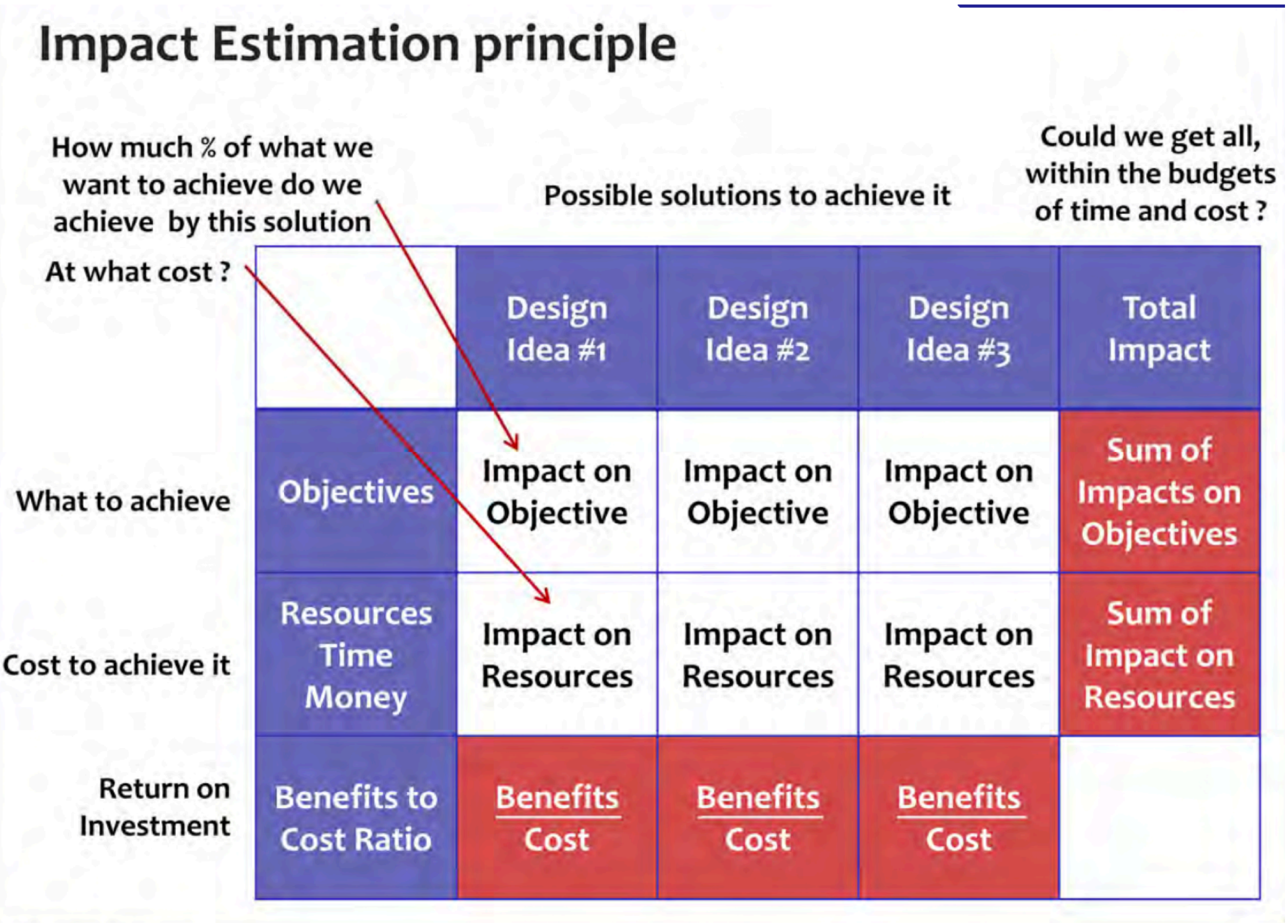
Part E. Quantification and Measurement

1.The entire Planguage method [2] is pervasively based on quantification, estimation and measurement of all variable system factors: i.e. Values, Resources.

2. In particular there is a well-developed tool ‘IET’ for understanding the causal relationship between any level of design, and any subsequent level of requirements. The Impact Estimation Table (IET) is the basic method for both estimation, and followup measurement.

3. The IET contains a number of devices to help us document and understand the qualities of estimates and measures (eg ± uncertainty, evidence, sources, credibility level, Meter)

4. The structure of defined Scales of measure, with any useful number of Scale Parameters (eg [Task], which each can have any useful number of attributes (eg Task = Start, Work On, Complete, Retract, Correct) permits extremely-detailed modelling, of complex systems, and consequent very-detailed levels of resolution ,of measurement of any value. or any resource.



Impact Estimation principle

How much % of what we want to achieve do we achieve by this solution

At what cost ?

Possible solutions to achieve it

Could we get all, within the budgets of time and cost ?

		Design Idea #1	Design Idea #2	Design Idea #3	Total Impact
What to achieve	Objectives	Impact on Objective	Impact on Objective	Impact on Objective	Sum of Impacts on Objectives
Cost to achieve it	Resources Time Money	Impact on Resources	Impact on Resources	Impact on Resources	Sum of Impact on Resources
Return on Investment	Benefits to Cost Ratio	$\frac{\text{Benefits}}{\text{Cost}}$	$\frac{\text{Benefits}}{\text{Cost}}$	$\frac{\text{Benefits}}{\text{Cost}}$	

Figure E: The basic structure of an Impact Estimation Table (Source Niels Malotaux, Ryan Shriver)

The basic ‘numerics’ for design of Planguage are:

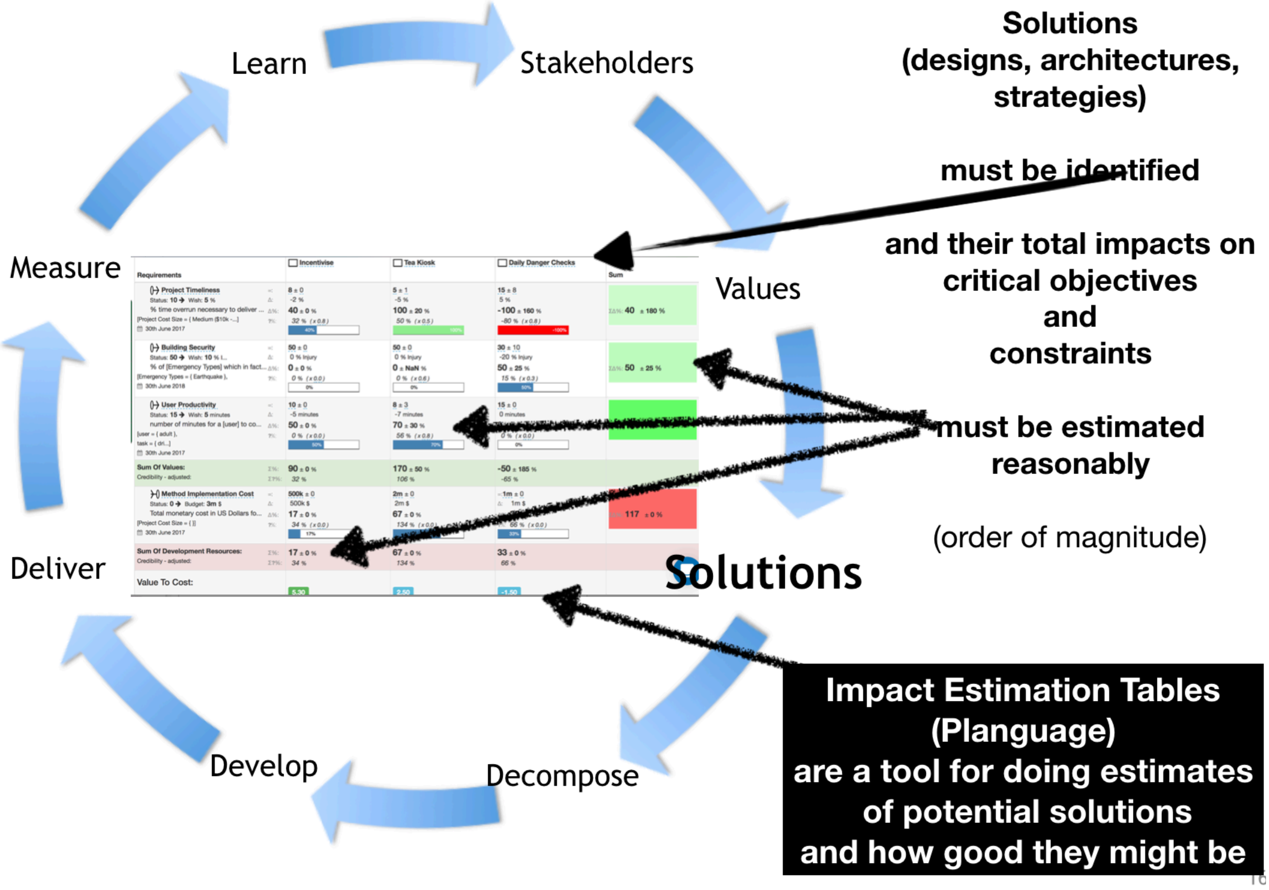
1. Quantification of all values and resources requirements, using a formally defined Scale of Measure. This is the foundation for all the other ‘numerics’.

2. Estimation of the **impacts expected from any design hypothesis**, on any and all quantified requirements (re-using their define Scale)

a. An estimate of impact along the scale measure, from a Status Level point.
b. A calculation of the % of way to a defined Target or Constraint level, on time.
c. A ± uncertainty, or spread of experience is estimated.
d. A Credibility Level (0.0 None, to 1.0 Perfect) based on evidence for estimate, and the source.
e. Several other derived numbers as in Figure E. above, and more.

3. The notion of a Meter, a defined process of measurement, to be specified in conjunction with the specified Scale of measure. Any number of Meters, designed with various qualities and costs, may be pinned as necessary.

4. The notion of numeric feedback, via the Meter measurements, at any Evo value delivery step: a measurement of reality, which can be compared with estimated impacts. The difference is analyzed the see if action, such as re-design, is needed [11].



Requirements		☐ Incentivise	☐ Tea Kiosk	☐ Daily Danger Checks	Sum
⟨⟩ Project Timeliness Status: 10 → Wish: 5 % % time overrun necessary to deliver ... [Project Cost Size = { Medium ($10k -...] 30th June 2017	=: Δ: Δ%: ?%:	8 ± 0 -2 % 40 ± 0 % 32 % (x 0.8) 40%	5 ± 1 -5 % 100 ± 20 % 50 % (x 0.5) 100%	15 ± 8 5 % -100 ± 160 % -80 % (x 0.8) -100%	ΣΔ%: 40 ± 180 %
⟨⟩ Building Security Status: 50 → Wish: 10 % I... % of [Emergency Types] which in fact... [Emergency Types = { Earthquake }, 30th June 2018	=: Δ: Δ%: ?%:	50 ± 0 0 % Injury 0 ± 0 % 0 % (x 0.0) 0%	50 ± 0 0 % Injury 0 ± NaN % 0 % (x 0.6) 0%	30 ± 10 -20 % Injury 50 ± 25 % 15 % (x 0.3) 50%	ΣΔ%: 50 ± 25 %
⟨⟩ User Productivity Status: 15 → Wish: 5 minutes number of minutes for a [user] to co... [user = { adult }, task = { dri...] 30th June 2017	=: Δ: Δ%: ?%:	10 ± 0 -5 minutes 50 ± 0 % 0 % (x 0.0) 50%	8 ± 3 -7 minutes 70 ± 30 % 56 % (x 0.8) 70%	15 ± 0 0 minutes 0 % (x 0.0) 0%	
Sum Of Values: Credibility - adjusted:	Σ%: Σ?%:	90 ± 0 % 32 %	170 ± 50 % 106 %	-50 ± 185 % -65 %	
⟩⟨ Method Implementation Cost Status: 0 → Budget: 3m $ Total monetary cost in US Dollars fo... [Project Cost Size = { }] 30th June 2017	=: Δ: Δ%: ?%:	500k ± 0 500k $ 17 ± 0 % 34 % (x 0.0) 17%	2m ± 0 2m $ 67 ± 0 % 134 % (x 0.0)	1m ± 0 1m $ 66 % (x 0.0) 33%	ΣΔ%: 117 ± 0 %
Sum Of Development Resources: Credibility - adjusted:	Σ%: Σ?%:	17 ± 0 % 34 %	67 ± 0 % 134 %	33 ± 0 % 66 %	
Value To Cost:		5.30	2.50	-1.50	

Solutions (designs, architectures, strategies)

must be identified

and their total impacts on critical objectives and constraints

must be estimated reasonably

(order of magnitude)

Impact Estimation Tables (Planguage) are a tool for doing estimates of potential solutions and how good they might be

The Gilb Evo Cycle of managing design through numeric data. [17C].

These are the numeric basics, from which we can perform a large number of analytical and presentation tasks: such as,

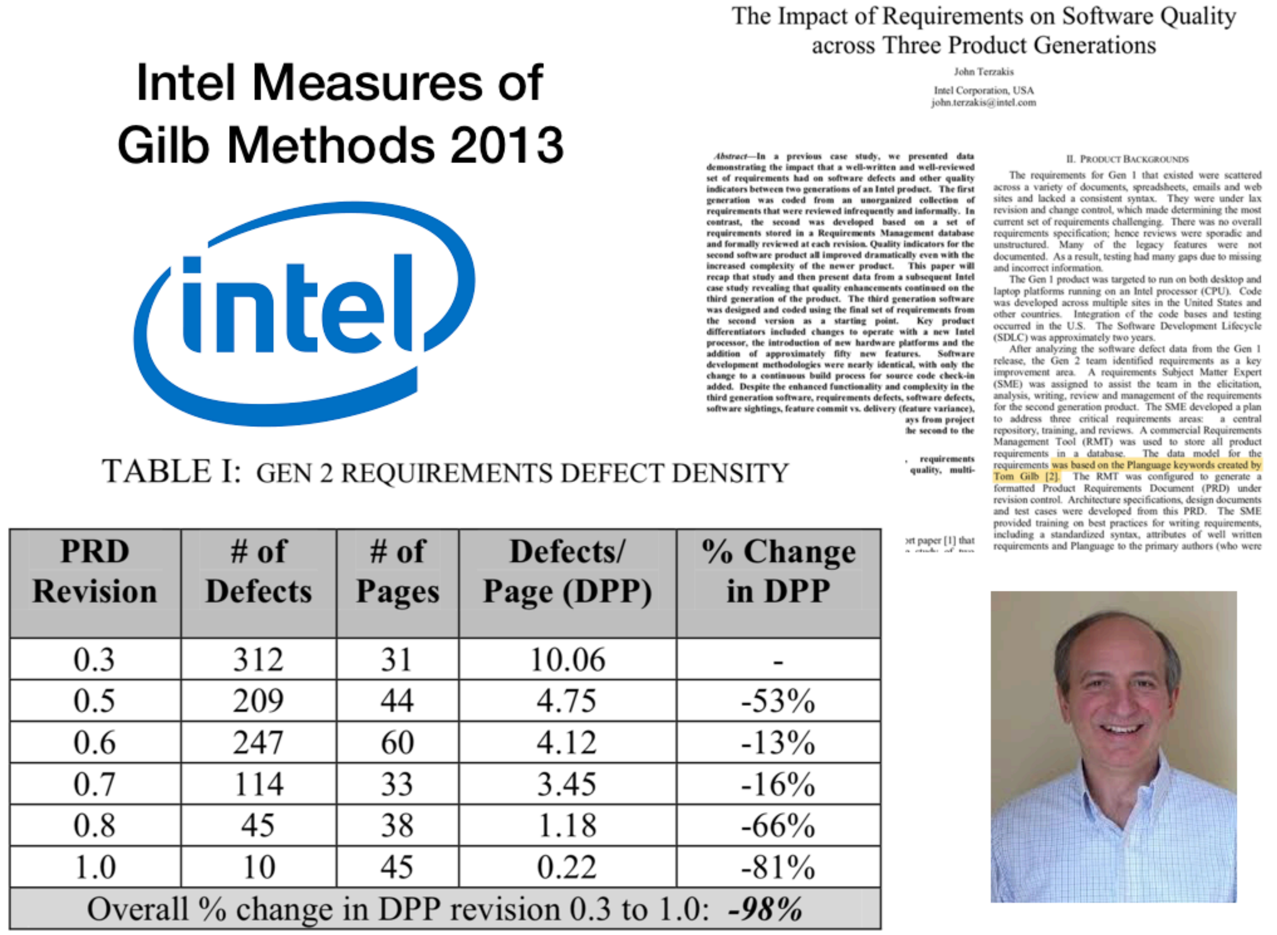
1. Detect inconsistencies (like too much budget used already for the designs)
2. Sort best designs into priority sequences, to manage the flow of value deliveries optimally.
3. Present bar diagrams showing cumulated values and costs, with risks and uncertainties
4. Compute the best priority designs, based on a chosen priority policy.
5. Update complex models of estimates based on small changes, to see overall effects.
6. Track cumulative value delivery and cumulative resource uses, to help manage projects.

7. Understand design risks in depth, for each individual design, and for sets of designs

Part F. Case studies: Experiential Validation.

In addition to these above measurement (Past E) devices, built into Planguage, we have several decades of published research, in Industry Publications, MSc, and PhD [22] level dissertations [19 HP, 20 Intel], regarding the observed industrial attributes of Planguage and related processes (Evo, SQC, CE [2]).

One example of method measurement is Terzakis [4, 2013], Intel studying requirements written in Planguage, and quality controlled using Spec QC, with a reported 233% Engineering productivity increase. Over 21,000 Intel engineers have been trained in these methods at Intel over 16+ Years.



Intel Measures of Gilb Methods 2013

The Impact of Requirements on Software Quality across Three Product Generations

John Terzakis

Intel Corporation, USA
john.terzakis@intel.com

Abstract—In a previous case study, we presented data demonstrating the impact that a well-written and well-reviewed set of requirements had on software defects and other quality indicators between two generations of an Intel product. The first generation was coded from an unorganized collection of requirements that were reviewed infrequently and informally. In contrast, the second was developed based on a set of requirements stored in a Requirements Management database and formally reviewed at each revision. Quality indicators for the second software product all improved dramatically even with the increased complexity of the newer product. This paper will recap that study and then present data from a subsequent Intel case study revealing that quality enhancements continued on the third generation of the product. The third generation software was designed and coded using the final set of requirements from the second version as a starting point. Key product differentiators included changes to operate with a new Intel processor, the introduction of new hardware platforms and the addition of approximately fifty new features. Software development methodologies were nearly identical, with only the change to a continuous build process for source code check-in added. Despite the enhanced functionality and complexity in the third generation software, requirements defects, software defects, software sightings, feature commit vs. delivery (feature variance),
ays from project
he second to the

, requirements
quality, multi-

ort paper [1] that

II. Product Backgrounds

The requirements for Gen 1 that existed were scattered across a variety of documents, spreadsheets, emails and web sites and lacked a consistent syntax. They were under lax revision and change control, which made determining the most current set of requirements challenging. There was no overall requirements specification; hence reviews were sporadic and unstructured. Many of the legacy features were not documented. As a result, testing had many gaps due to missing and incorrect information.

The Gen 1 product was targeted to run on both desktop and laptop platforms running on an Intel processor (CPU). Code was developed across multiple sites in the United States and other countries. Integration of the code bases and testing occurred in the U.S. The Software Development Lifecycle (SDLC) was approximately two years.

After analyzing the software defect data from the Gen 1 release, the Gen 2 team identified requirements as a key improvement area. A requirements Subject Matter Expert (SME) was assigned to assist the team in the elicitation, analysis, writing, review and management of the requirements for the second generation product. The SME developed a plan to address three critical requirements areas: a central repository, training, and reviews. A commercial Requirements Management Tool (RMT) was used to store all product requirements in a database. The data model for the requirements was based on the Planguage keywords created by Tom Gilb [2]. The RMT was configured to generate a formatted Product Requirements Document (PRD) under revision control. Architecture specifications, design documents and test cases were developed from this PRD. The SME provided training on best practices for writing requirements, including a standardized syntax, attributes of well written requirements and Planguage to the primary authors (who were

TABLE I: GEN 2 REQUIREMENTS DEFECT DENSITY

PRD Revision	# of Defects	# of Pages	Defects/ Page (DPP)	% Change in DPP
0.3	312	31	10.06	-
0.5	209	44	4.75	-53%
0.6	247	60	4.12	-13%
0.7	114	33	3.45	-16%
0.8	45	38	1.18	-66%
1.0	10	45	0.22	-81%
Overall % change in DPP revision 0.3 to 1.0: ***-98%***				

Figure F. Terzakis, Intel, 2013 [20C] Industrial experience with 'Planguage for requirements' (chip design) and 'Spec QC' to measure specification quality before releasing it downstream. Initial submissions are at least 10X less defective than without using Planguage (same experience measure at Citigroup), 10 DDP with Planguage vs. over 100 DPP without). But the measurement process (Spec QC) motivates the requirements team to get 50X better, upstream. One way of looking at this is that Planguage + SQC is measurably 50x10 = 500x better quality.

Erik Simmons, who led the Intel implementation effort for almost 2 decades has written about his experiences, the short summary he says [2] 'This stuff works'. [20B, detailed sides report]

The IEEE (Sarah Gregory, Intel and IEEE) has selected (Best Known Practice) Planguage for its coming requirements Training Courses. 2019 in process, not released yet.

The Planguage/Evo/SQC case studies, exceed 100, and are spread out over books [2, 3, 5 and many more], free papers and slides [http://concepts.gilb.com/file24], and generally available from www.Gilb.com, or from the Internet. The case studies begin in the late 1960s as conference papers.

*** The following sections: Part G and on are organized by the topics in the table, so as to argue th case for a Design Theory in that manner.**

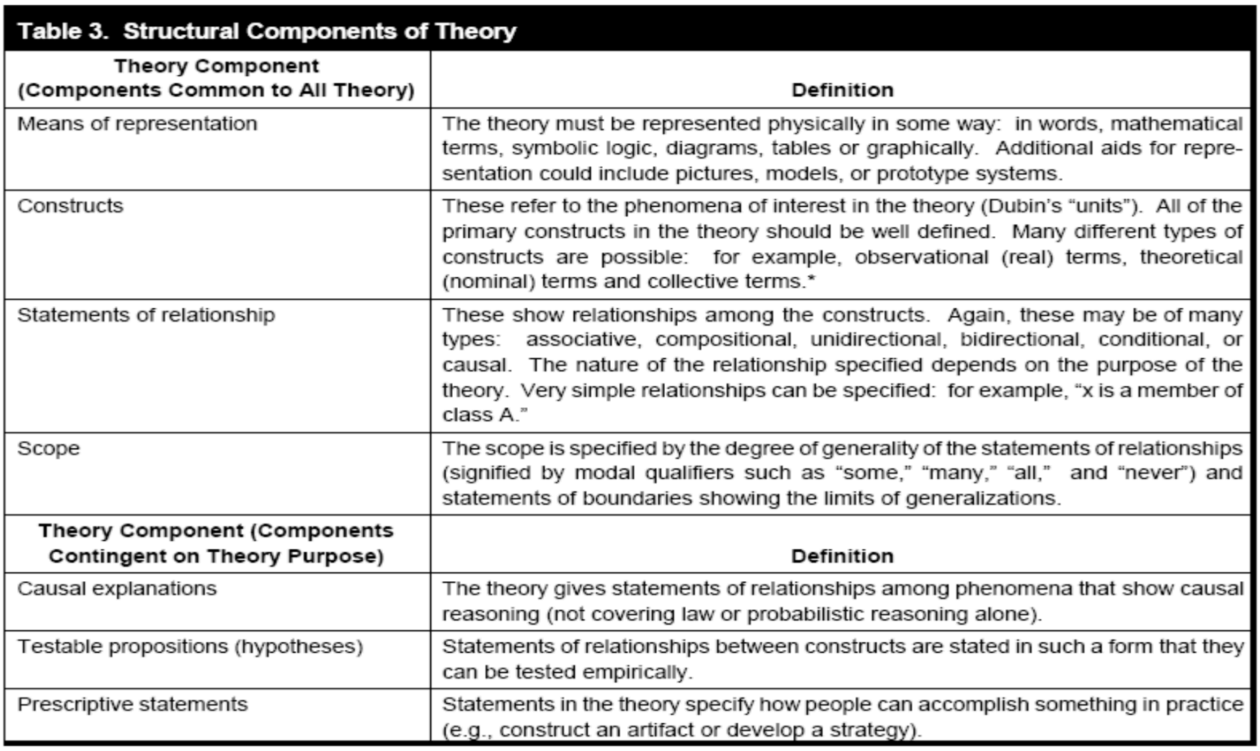


Table 3. Structural Components of Theory

Theory Component (Components Common to All Theory)	Definition
Means of representation	The theory must be represented physically in some way: in words, mathematical terms, symbolic logic, diagrams, tables or graphically. Additional aids for representation could include pictures, models, or prototype systems.
Constructs	These refer to the phenomena of interest in the theory (Dubin's "units"). All of the primary constructs in the theory should be well defined. Many different types of constructs are possible: for example, observational (real) terms, theoretical (nominal) terms and collective terms.*
Statements of relationship	These show relationships among the constructs. Again, these may be of many types: associative, compositional, unidirectional, bidirectional, conditional, or causal. The nature of the relationship specified depends on the purpose of the theory. Very simple relationships can be specified: for example, "x is a member of class A."
Scope	The scope is specified by the degree of generality of the statements of relationships (signified by modal qualifiers such as "some," "many," "all," and "never") and statements of boundaries showing the limits of generalizations.
Theory Component (Components Contingent on Theory Purpose)	**Definition**
Causal explanations	The theory gives statements of relationships among phenomena that show causal reasoning (not covering law or probabilistic reasoning alone).
Testable propositions (hypotheses)	Statements of relationships between constructs are stated in such a form that they can be tested empirically.
Prescriptive statements	Statements in the theory specify how people can accomplish something in practice (e.g., construct an artifact or develop a strategy).

Figure 3. Structural Components of Theory (adopted from Gregor, 2006)

Part G. ‘Means of Representation’ <- Table 3

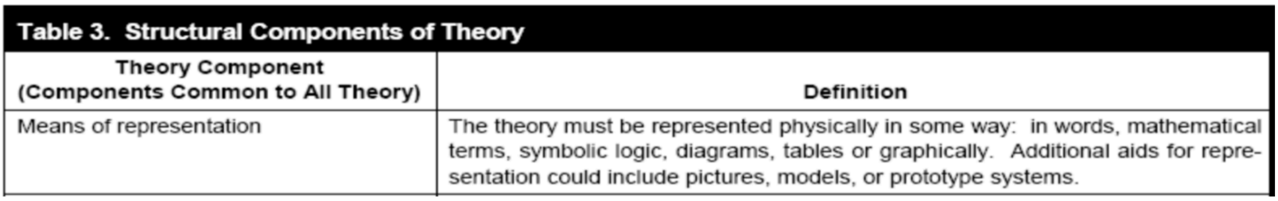


Table 3. Structural Components of Theory

Theory Component (Components Common to All Theory)	Definition
Means of representation	The theory must be represented physically in some way: in words, mathematical terms, symbolic logic, diagrams, tables or graphically. Additional aids for representation could include pictures, models, or prototype systems.

Planguage Designed for Computer Intelligibility

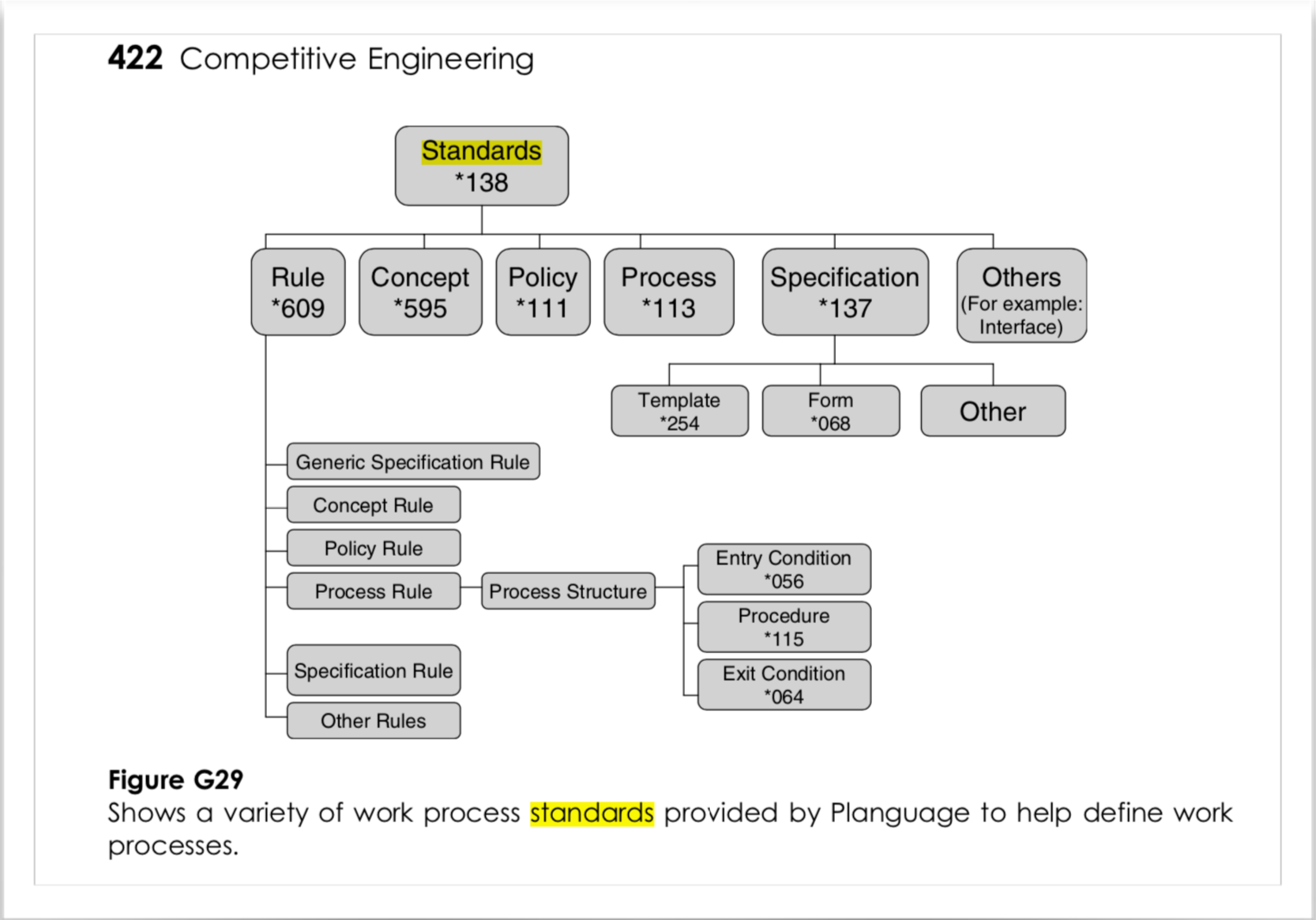
Planguage has always been developed with computer intelligibility in mind, and many tools to support it have been made [8].

Concept Glossary

The key to this is the well-developed Planguage Concept Glossary [2] where not only are design and planning concepts well-defined, and stable; but concepts are ‘well aware’ of all of the other concepts, both in the Planguage Concept Glossary, and in any project-related, or organizational-related extensions of the concepts. Absolutely all concepts, and user definitions (design objects for example) have unique hierarchical ‘Tags’. We mandate (Planguage Rules) reuse of concepts from a ‘master version’.

Planguage Standards

Planguage has a large number of ‘Standards’ defining it [2], among them ‘Rules’ (for best practice specification) and Design Processes. In addition there are well-‘defined Entry and Exit Conditions to all processes. These enable both human understanding consistently, and computer application automation.



422 Competitive Engineering

Figure G29

Shows a variety of work process standards provided by Planguage to help define work processes.

Plicons: Symbolic Notation

Many Planguage Concepts can be accessed from a defined set of symbolic notation [7].
These are designed in 2 formats: Drawn Icons, and Keyed Icons. Which are reasonably consistent with each other.

This idea was originally inspired by Blisssymbolics/Semantography (a *universal* language) but Plicons are a restricted set, to the domain of *planning and designing systems*.

These Plicons are used in *teaching* concepts (very useful for teaching theory of design!), in books [2] and in the tools [8]. There are numerous examples of keyed plicons used in this paper above.

I have attempted to define Planguage concepts (ideally the whole Planguage) by using the Plicons as a language. But this needs more work, and has more ‘scientific and theory’ interest, than my daily practical ‘design project’ use. So it awaits ‘academic development’. But it is potentially a form of systems mathematical notation. A great subject for a PhD in my opinion.

Digital Planguage Standards

The Planguage tools, particularly ValPlan.net, essentially build a *‘digital project database’* containing a large number of design and requirements specifications, and ‘background information (like who are the stakeholders, and spec owners for a design).

All Planguage tools, many unofficial and client private tools, necessarily embed the standards of Planguage. valplan.net to built in detail on the Competitive Engineering [2] framework, and contains many of the standards, such as Glossary, Rules, Processes, Templates, and Policies (such as design prioritization) very directly embedded.

The consequence of the ‘digital project database’ is:

1. Any interesting visual and text displays can be generated for project presentation and analysis, very selectively and quite visually. [3, 5, 8]

2. A great deal of automated quality control of the project design plans is possible.
Some of this is implemented, but we have formally designed (in the ValPlan tool of course) a great deal more, awaiting our capacity to implemen*t.*

EDIT NOTE: it would be possible to illustrate these things using outputs from ValPlan.net but they are richly illustrated in my books and writings already (and a few examples above).You can of course got to ValPlan.net and see an overview of some of this right now.

Gibson, Bechtel, ‘graphMetrix’ [10]

In 2015, Fredrik Gibson, a formally trained architect, working at Bechtel in San Francisco, contacted me. He said he had read my Competitive Engineering [2] book 3 times, and found that Planguage was a good framework for his building advanced AI tools for the *construction industry.*

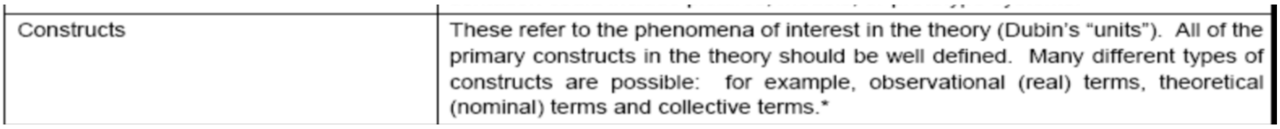
In fact he had **already built** tools using Planguage ideas, at Bechtel. And he demonstrated them for me over the internet. But these were for Bechtel internal use.

‘You could have blown me over with a feather’, as they say. This came ‘out of the blue’, quite unexpected. But it did demonstrate the universal design application of Planguage ideas.

Today, (June 2019) Gibson has built his own startup, graphMetrix, and is far advanced with this Construction Industry tool building. Kai Gilb is very active in furthering the relationship. Go to the site [10] and be amazed.

Part H. Constructs*

I suspect this section, Constructs, is covered above in several places: Primarily by referring to the Planguage Concept Glossary (point 2 Above)



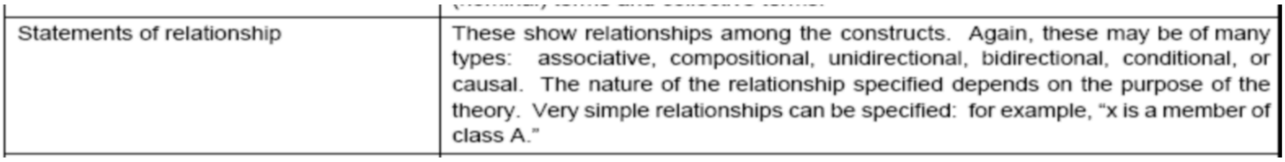
Constructs	These refer to the phenomena of interest in the theory (Dubin's "units"). All of the primary constructs in the theory should be well defined. Many different types of constructs are possible: for example, observational (real) terms, theoretical (nominal) terms and collective terms.*

Part I. Statements of Relationship*

There are a large quantity of relationships defined in Planguage.

Some of these are generic and static: they belong to the Planguage Concept Glossary definitions. For example 'Stakeholders have Values'.

Others are specific relationships defined in a projects: for example: The Stakeholder for Value X, are S1, S2, and S3.



Statements of relationship	These show relationships among the constructs. Again, these may be of many types: associative, compositional, unidirectional, bidirectional, conditional, or causal. The nature of the relationship specified depends on the purpose of the theory. Very simple relationships can be specified: for example, "x is a member of class A."

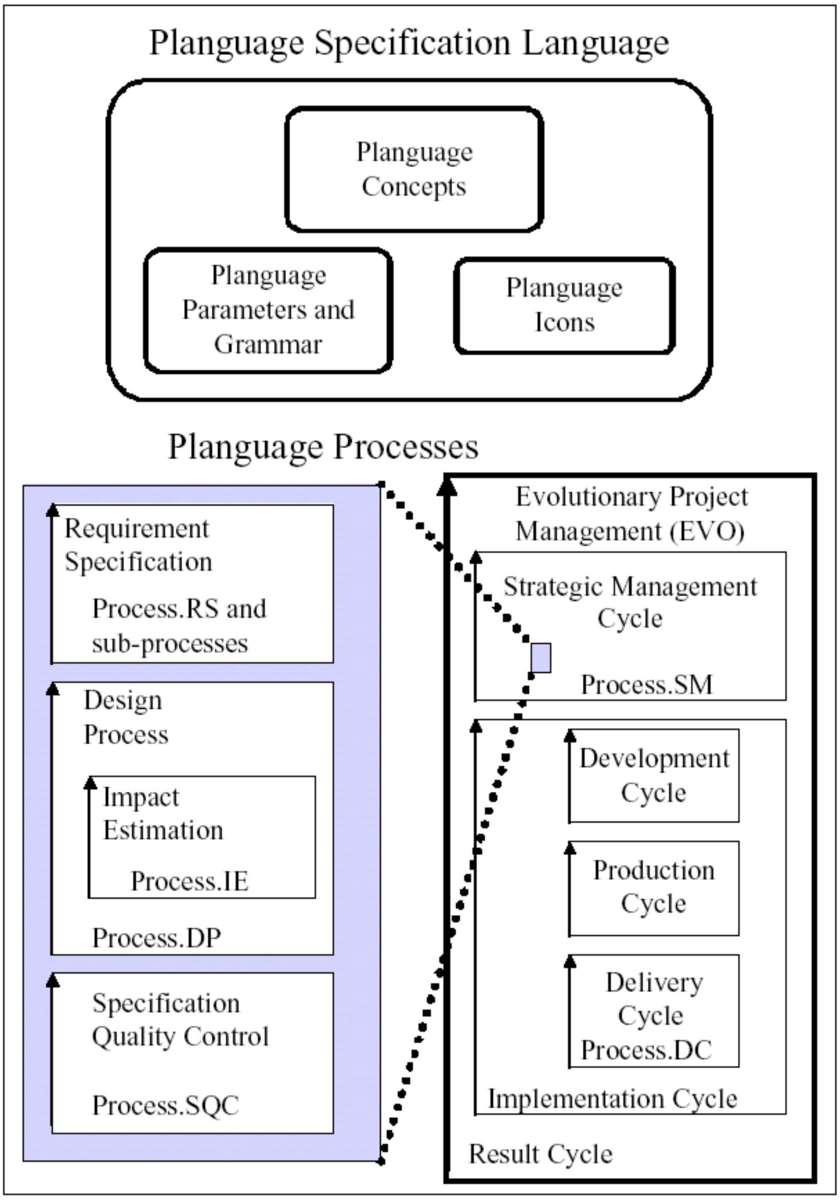


Figure I A. Planguage is a combination of defined languages (words, icons, numbers), and defined engineering processes.

Planguage has a rich set of graphical symbols and about 685 formally defined concepts.

It is based on well over 100 basic principles [2, 9, 18].

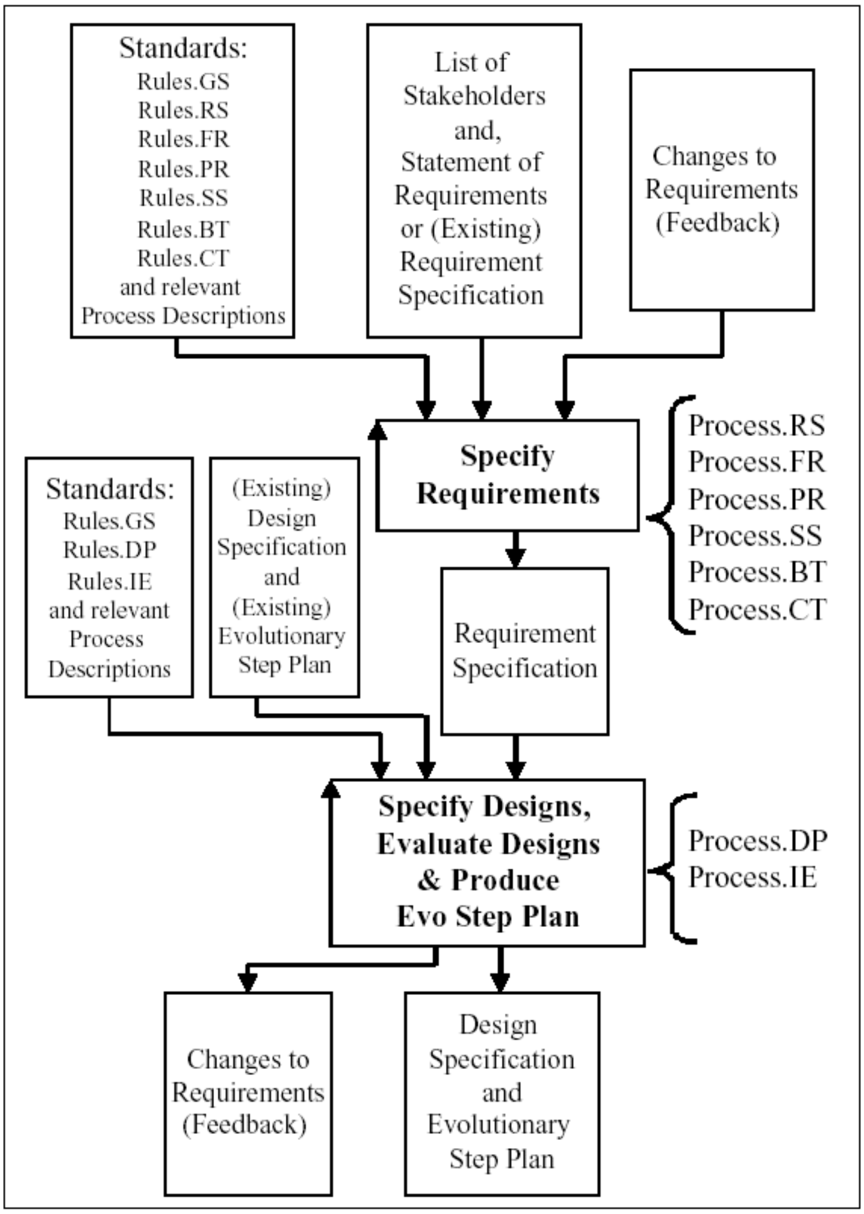

Standards:
Rules.GS
Rules.RS
Rules.FR
Rules.PR
Rules.SS
Rules.BT
Rules.CT
and relevant
Process Descriptions
List of Stakeholders and, Statement of Requirements or (Existing) Requirement Specification
Changes to Requirements (Feedback)
Specify Requirements
Process.RS
Process.FR
Process.PR
Process.SS
Process.BT
Process.CT
Standards:
Rules.GS
Rules.DP
Rules.IE
and relevant Process Descriptions
(Existing) Design Specification and (Existing) Evolutionary Step Plan
Requirement Specification
Specify Designs, Evaluate Designs & Produce Evo Step Plan
Process.DP
Process.IE
Changes to Requirements (Feedback)
Design Specification and Evolutionary Step Plan

Figure I B. Standards: Planguage is based on well-defined processes, and on well-defined 'Rules' for specification.

These Planguage processes and rules all strongly support the management of quantified qualities and other values, in relation to budgeted resources. That is what we describe as an engineering process.

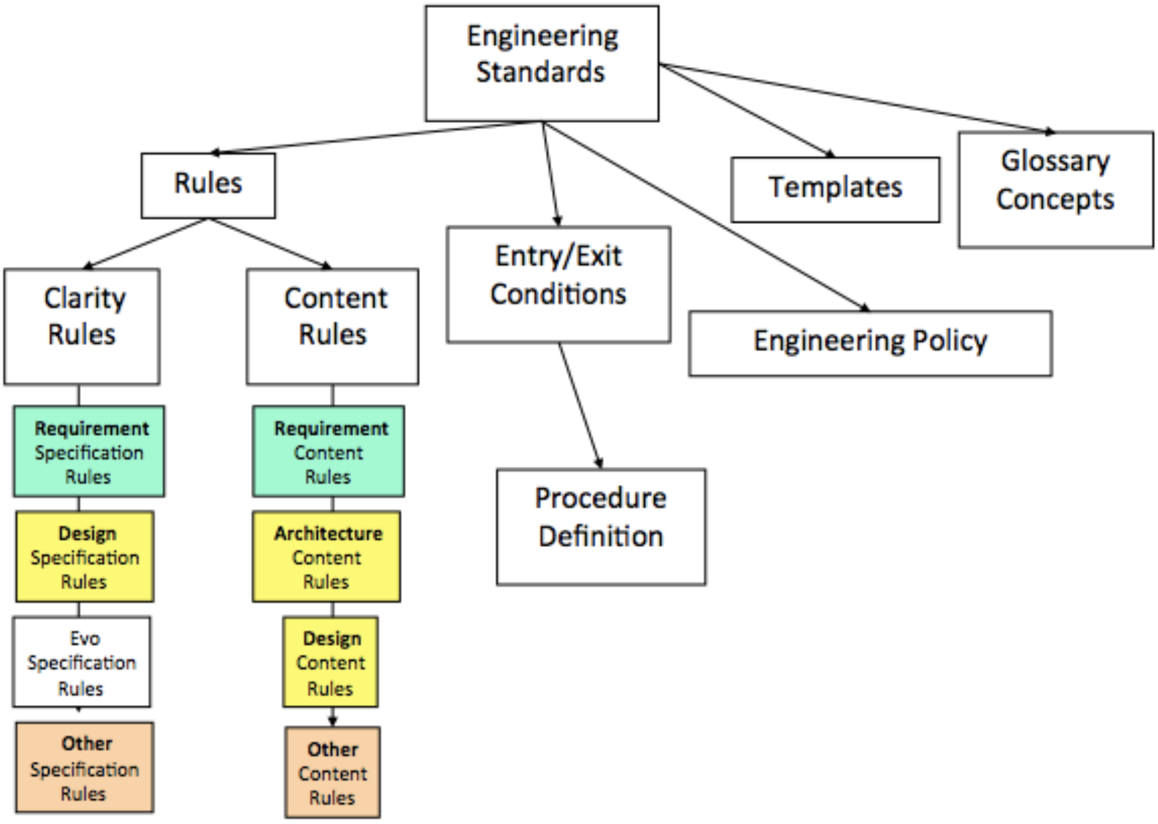


Figure I C. : Planguage has a wide variety of engineering standards.

One interesting detail is that we have clearly distinguished between *clarity* of engineering specification and *content*. For example we have rules that values and qualities must be expressed quantitatively (clarity). But it also has rules that say these same quantified qualities should carefully distinguish between a target (value level to achieve), and a constraint (a minimum level for survival or partial payment) - content.

Planguage also suggests a number of engineering management **policies**; such as estimation of the 'value and cost' impacts, of all design hypothesis suggestions.

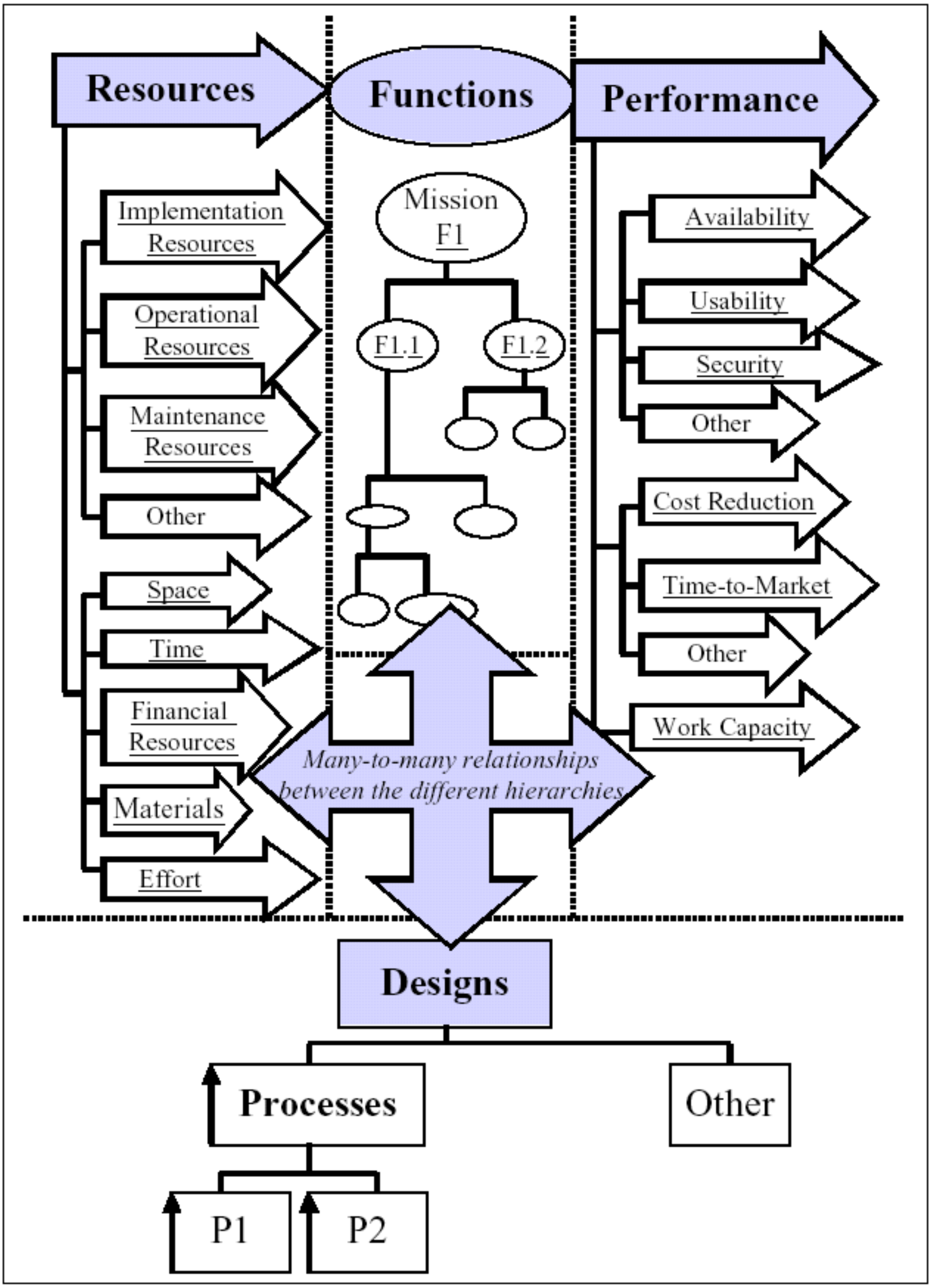


Figure I D: Planguage is tuned to the real-world complexity of many-to-many relationships.

Planguage handles many levels of concern, multiple improvement objectives, for multiple stakeholders, multiple resource constraints, multiple functions, multiple designs, and multiple functions all in one integrated planning language.

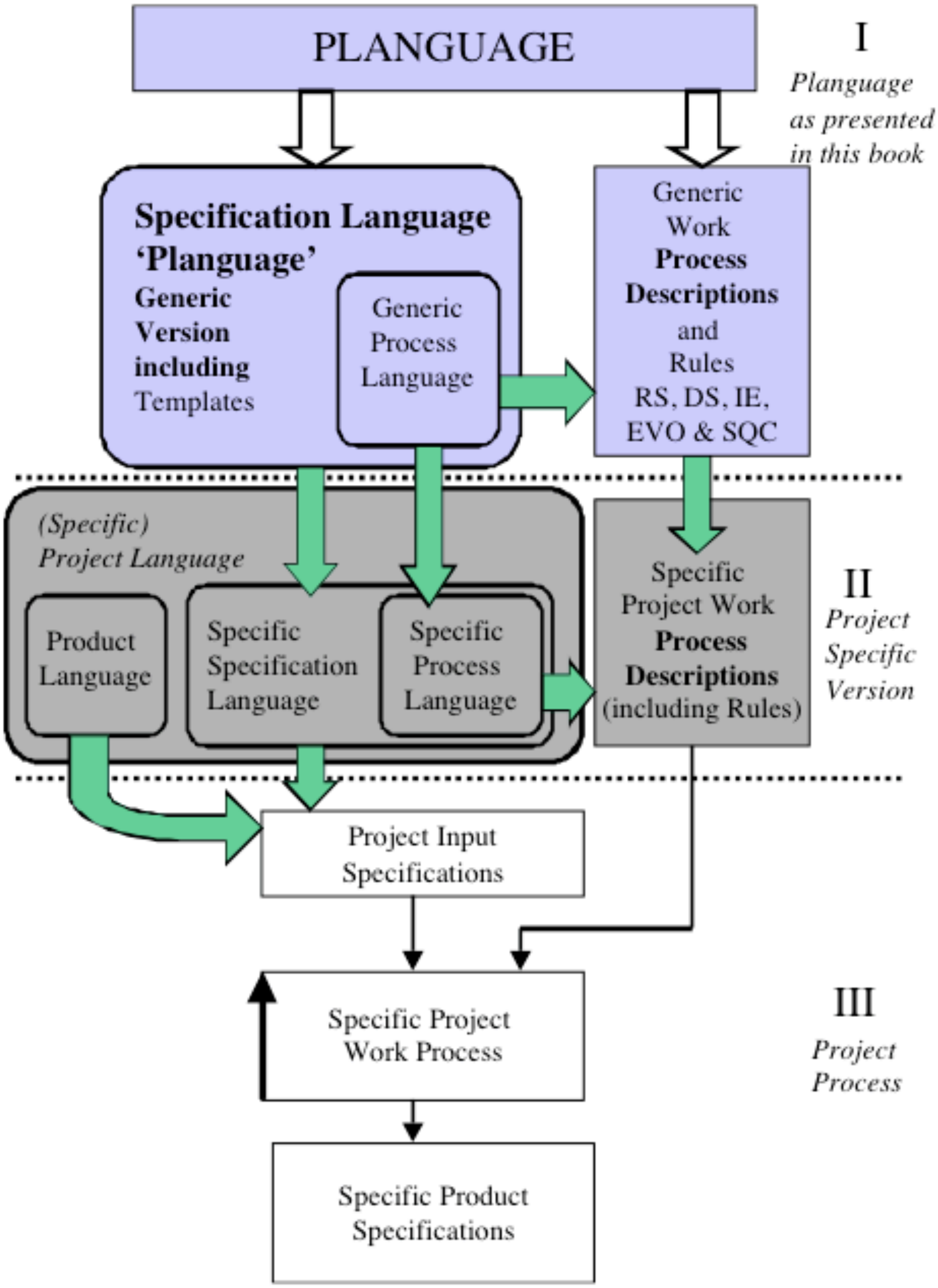
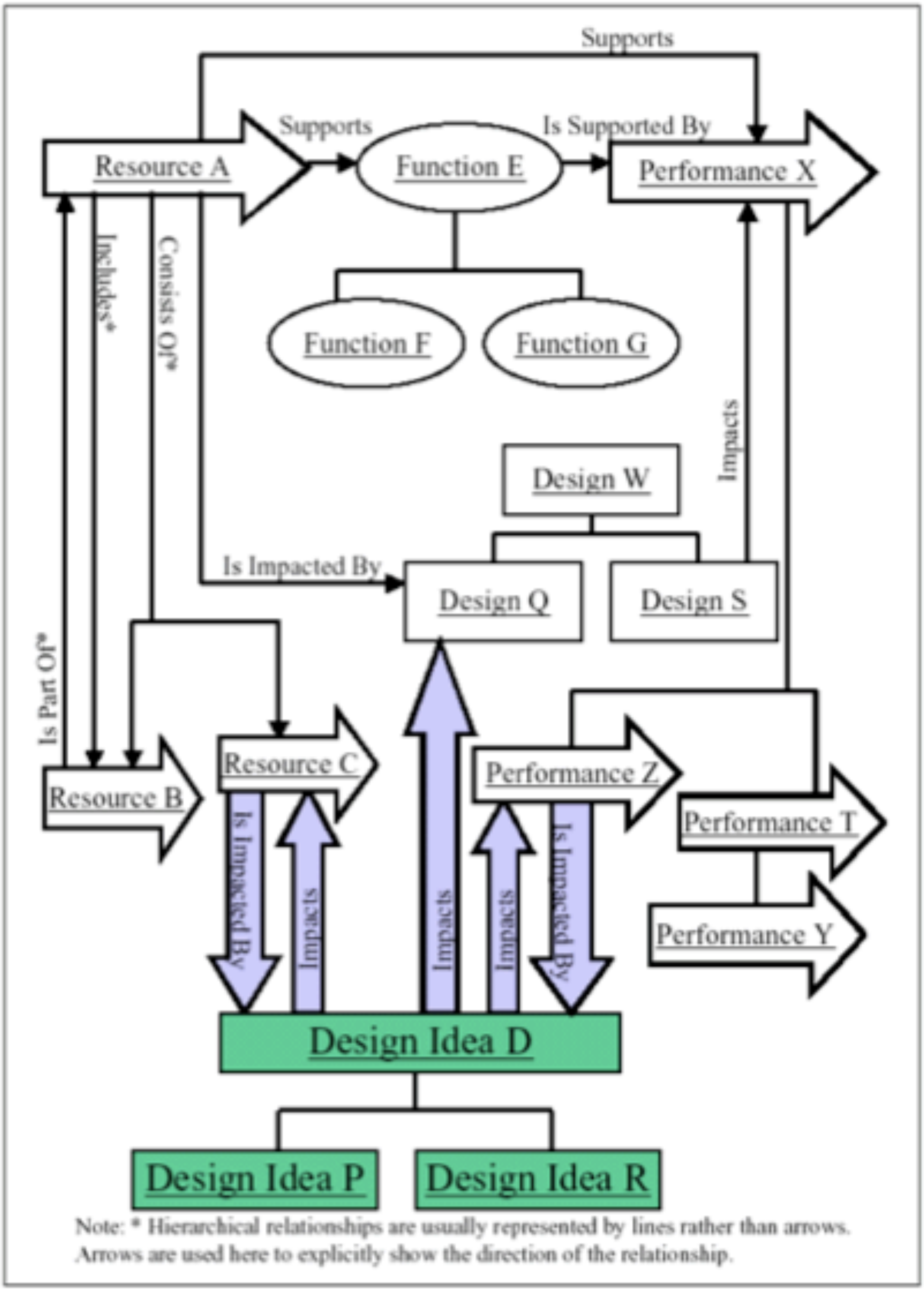

PLANGUAGE
I
Planguage as presented in this book
Specification Language 'Planguage'
Generic Version including Templates
Generic Process Language
Generic Work Process Descriptions and Rules RS, DS, IE, EVO & SQC
(Specific) Project Language
Product Language
Specific Specification Language
Specific Process Language
Specific Project Work Process Descriptions (including Rules)
II
Project Specific Version
Project Input Specifications
Specific Project Work Process
III
Project Process
Specific Product Specifications

Figure I E. Planguage operates at different levels of abstraction, a generic Planguage level, a project level; and both of these have their special components.



Note: * Hierarchical relationships are usually represented by lines rather than arrows. Arrows are used here to explicitly show the direction of the relationship.

Figure I F. Planguage supports a very large number of specific relationships between planning elements (design, performance, resources, function, and constraints) .

This very pervasive use of pointers to related system components helps in change management, and risk management. You could go as far as saying that almost all Planguage statements express some kind of relationship to other components of the system planning.

Som of my friends dislike this diagram, but I love it!

Here are some examples of relationship parameters in Planguage:

Authority
Source
Owner
Author
Implementer
Impacts
Supports
Supported By
Version
Derived From
Sub-component of
Sub-components {list}
Dependencies
Contract
Test Case
Scenario
Model
And more!

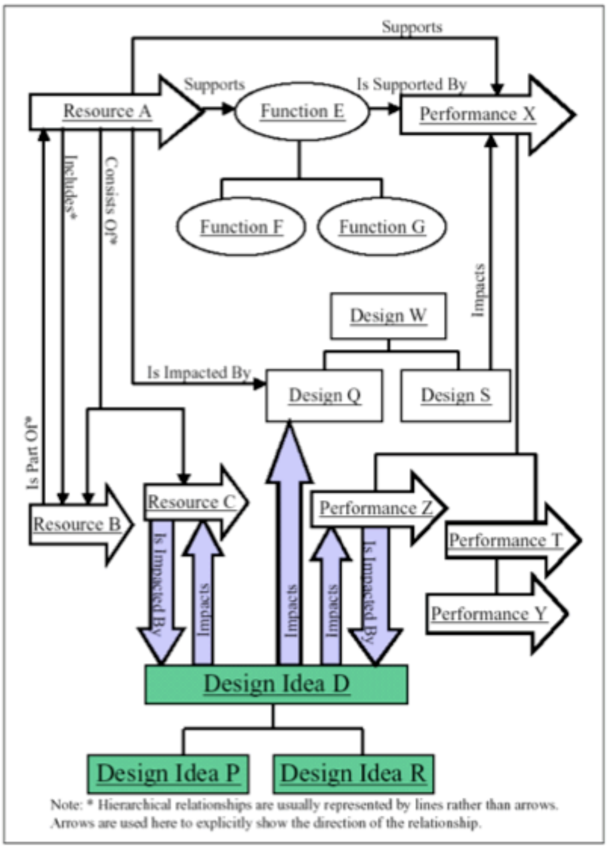


Figure IG. A sample of Planguage relationship parameters.

Part J. Scope*

Scope	The scope is specified by the degree of generality of the statements of relationships (signified by modal qualifiers such as "some," "many," "all," and "never") and statements of boundaries showing the limits of generalizations.

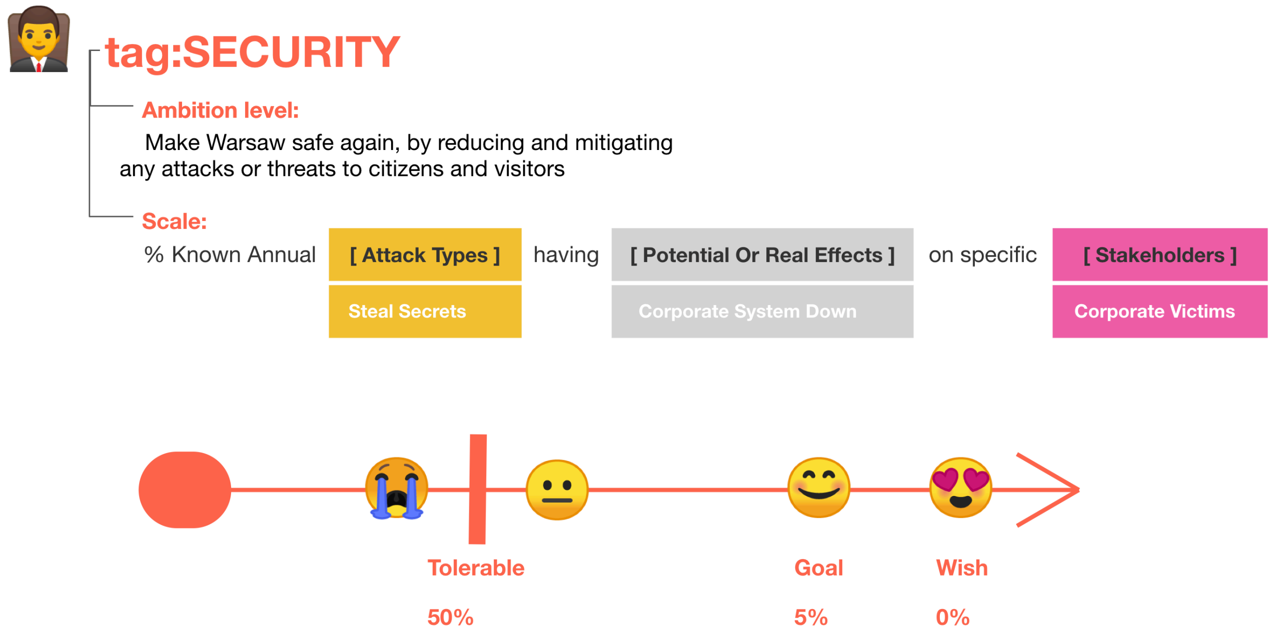
The **Scope of Planguage** is

Any planning of any future.
For any 'system'.
Real world measured feedback versus plans, in order to control processes or projects towards objectives, within constraint.

I am unsure as to how to handle thisScope subject. Maybe it is covered in all the above descriptions.

But, There are very many scope-defining tools in Planguage, as detailed above, and in the references below, like Value Planning [5]

However I would like to refer to one detail, among many, already described several times: the [Scale Parameter] definition. It is a powerful tool for defining the exact multidimensional scope of a requirement, and thus also a design's impact on the defined scope of that requirement.



tag:SECURITY

Ambition level:
Make Warsaw safe again, by reducing and mitigating any attacks or threats to citizens and visitors

Scale:
% Known Annual [Attack Types] having [Potential Or Real Effects] on specific [Stakeholders]

Steal Secrets | Corporate System Down | Corporate Victims

Tolerable 50%

Goal 5%

Wish 0%

A good second place is our extensive analysis, and integration in all requirements, of the vast **stakeholder** colony, showing relations to stakeholders explicitly.

To manage your ecosystem choose critical values (Top 10) which can kill your project

* Theory Components (Gregor, 2006, [1])

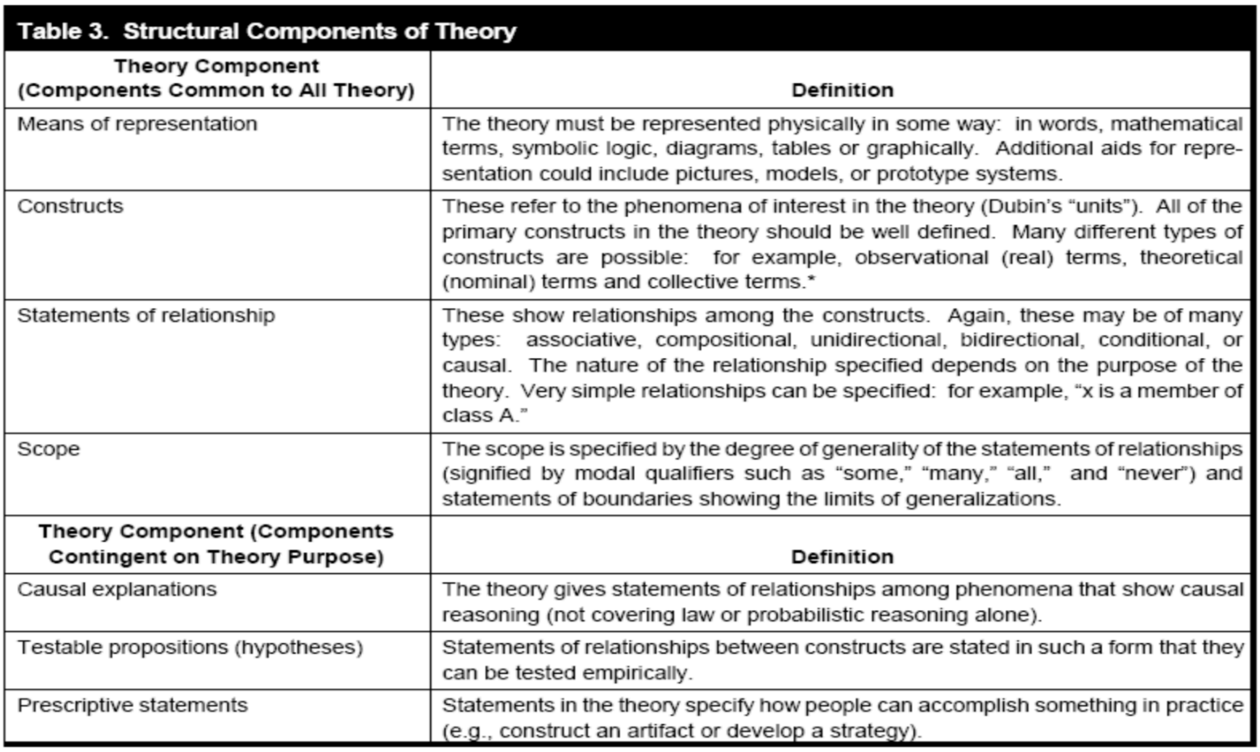
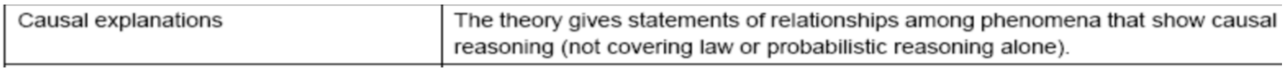


Table 3. Structural Components of Theory

Theory Component (Components Common to All Theory)	Definition
Means of representation	The theory must be represented physically in some way: in words, mathematical terms, symbolic logic, diagrams, tables or graphically. Additional aids for representation could include pictures, models, or prototype systems.
Constructs	These refer to the phenomena of interest in the theory (Dubin's "units"). All of the primary constructs in the theory should be well defined. Many different types of constructs are possible: for example, observational (real) terms, theoretical (nominal) terms and collective terms.*
Statements of relationship	These show relationships among the constructs. Again, these may be of many types: associative, compositional, unidirectional, bidirectional, conditional, or causal. The nature of the relationship specified depends on the purpose of the theory. Very simple relationships can be specified: for example, "x is a member of class A."
Scope	The scope is specified by the degree of generality of the statements of relationships (signified by modal qualifiers such as "some," "many," "all," and "never") and statements of boundaries showing the limits of generalizations.
Theory Component (Components Contingent on Theory Purpose)	**Definition**
Causal explanations	The theory gives statements of relationships among phenomena that show causal reasoning (not covering law or probabilistic reasoning alone).
Testable propositions (hypotheses)	Statements of relationships between constructs are stated in such a form that they can be tested empirically.
Prescriptive statements	Statements in the theory specify how people can accomplish something in practice (e.g., construct an artifact or develop a strategy).

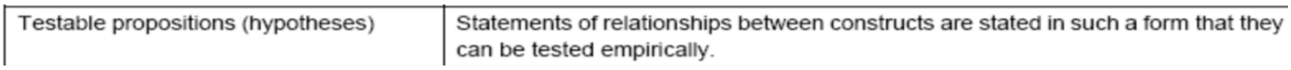
Figure 3. *Structural Components of Theory (adopted from Gregor, 2006)*

Part K. Causal Explanations*



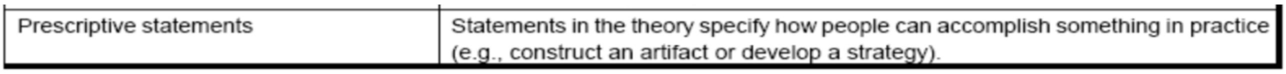
Causal explanations	The theory gives statements of relationships among phenomena that show causal reasoning (not covering law or probabilistic reasoning alone).

Part L. Testable Propositions (Hypothesis)*



Testable propositions (hypotheses)	Statements of relationships between constructs are stated in such a form that they can be tested empirically.

Part M. Prescriptive Statements*



Prescriptive statements	Statements in the theory specify how people can accomplish something in practice (e.g., construct an artifact or develop a strategy).

References

1. WRITING MY NEXT DESIGN SCIENCE RESEARCH MASTERPIECE: BUT HOW DO I MAKE A THEORETICAL CONTRIBUTION TO DSR?
Samir Chatterjee, PhD

2. Gilb, T, Competitive Engineering, 2005
This is the primary 'standard' for Planguage and related methods (SQC, Evo)
It contains an edited subset (a third of whole book) of the Planguage Concept Glossary [12].

3. GILB, T. "TECHNOSCOPES", https://www.gilb.com/offers/YYAMFQBH/checkout
Digital book, 2018. 100 Tools for understanding Wicked and Complex Systems.

4. John Terzakis, "The impact of requirements on software quality across three product generations," [Intel, Industrial Study], Based on Gilb methods (Planguage and SQC)
2013 21st IEEE International Requirements Engineering Conference (RE), Rio de Janeiro, 2013, pp. 284-289.
https://www.thinkmind.org/download.php?articleid=iccgi_2013_3_10_10012

5. 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. It also contains more than 200 references to freely downloadable case studies, papers and slides.

6. Gilb. "SCALE-FREE:
Practical Scaling Methods for Industrial Systems Engineering"
lecture slides, http://concepts.gilb.com/dl892
2016, Considerable citation of Intel experience with Planguage method, by Erik Simmons.

Gilb "Beyond Scaling: Scale-free Principles for Agile Value Delivery - Agile Engineering"
http://www.gilb.com/dl865 (Paper)
(Jan 8 2016). This paper contains considerable detailed systemic explanation, theory, as to why the Planguage methods are 'Scale Free'. The scale free theory is backed up by among others the detailed E Simmons reports in the slides above. As well as the extensive range of project scales in the case studies.

7. T. Gilb, 'Plicons: A Graphic Planning Language for Systems Engineering'
(Plicons Paper)
http://www.gilb.com/DL37
This symbolic language is 'human language' independent, like musical notes or electrical symbols. I have actually attempted to define Planguage using these symbols, but this area needs further rigorous work for elegant development.

8. www.Valplan.net : An advanced planning tool (app) based on Planguage.

This is a publicly available (many tools were made for in house use only by our clients, example 2 levels of tools by Confirmit AS, 2003-2005), rich, tool based on Planguage as defined in Competitive Engineering [2] built by Richard Smith (UK).

Planguage was designed by Gilbs, with automation in mind. There have been a series of automated tools built for decades, starting about 1979 with the 'Aspect Engine' (reported in Gilb, Principles of Software Engineering Management, 1988), constructed by Prof. Lech Krzanik (Finland), for his PhD Thesis (Krakow, Poland). Kai Gilb built a predecessor tool from Spreadsheets, used in consulting and training practice, for decades (1993-2016), until superseded by 'Needsandmeanscoms' (now rebranded ValPlan.net) and marked by Kai Gilb. The tool has undergone about 4 years of field Beta-testing (about 2014-2019) before being released for commercial application (May 20 2019).

9. Gilb, Principles of Software Engineering Management, 1988

Internet Chapters (text only, no illustrations):
A. pdf 'Ch 14 POSEM Productivity' gilb.com/dl560
B. pdf 'Ch 15 POSEM Deeper Perspectives on Evolutionary delivery gilb.com/dl561

This includes a page extra of quotations from Agile Gurus crediting it as inspiration for them, and it being first.
C. Chapter 13.4 (page 237-241) Open Ended Architecture
D. Chapter 21 ICL Inspection Experiences

It contains about 144 stated principles. The hardcover book is still for sale, new and used.

10. Frederick Gibson, Graphmetrix. Building Industry Software based on Gilbs Competitive Engineering/Planguage Concepts, 2019

https://graphmetrix.com/web/

"Our technology links the Owner's Goals to Product Manufacturing, Engineering, Procurement, Construction and Operations"
Tom Gilb and Kai Gilb are on the Board of Advisors of Graphmetrix.
F. Gibson: Construction innovator for the last 8 years on multi-billion dollar projects at Bechtel, creator of Bechtel's iHub knowledge graph platform, principal architect at FGA+A for 14 years and former startup founder.

11. Cleanroom Case Study

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 on

The major insight and example of this reference is the clear experience that when the architect (Quinnan) reviews cost and value numbers at every (agile) delivery step, and makes new architecture decisions to improve costs and values, then this has the overall effect of guaranteeing on time, under budget high quality (military and space) software deliveries, year after year, without exception.

This is industrial verification of the **theory of incremental design** (agile design, if you will). I reported this experience in [9], and had numerous first hand meetings with Mills and his Cleanroom team. The method is essentially identical to my Competitive Engineering [2] 'Evo method. But in the IBM case, the incremental architecture practice is much better documented.

12. Planguage Concept Glossary:

12a: The private master glossary, updated by Tom Gilb, and periodically shared at gilb.com. I will share on email request the latest version to serious researchers.
http://www.gilb.com/DL386

12b Paper on the principles of the Concept Glossary.
"A Conceptual Glossary for Systems Engineering:
Define the Concept, don't quibble about the terms." Gilb. 2006
http://www.gilb.com/dl565

12c. Planguage Concept Glossary as edited in 'Competitive Engineering' [2] book 2005
http://www.gilb.com/DL387
This is about 10% of the detail of the full glossary (12a)

12d. Glossary at gilb.com website
http://concepts.gilb.com/A?structure=Glossary&page_ref_id=126
This was and is an open source collective effort by students and friends. If you want to help build it up, please volunteer (kai@Gilb.com has organized this)
It is mainly based in the CE book Glossary [2]

12e. Glossary built into the app www.ValPlan.net

The app glossary is at 2 levels, at least

1. **A short definition is displayed when the cursor is moved over a defined Planguage parameter (like 'Scale').**
2. **A full, Competitive Engineering Glossary Concept is displayed on request in a side bar, for the additional parameters which can be added to a specification window.**
3. **Many pragmatic concepts are defined or explained, as they come up, for example 'Roles of Stakeholders'. But not all of these are official Planguage. Some are defined on the fly as we expand the app by others than Tom Gilb. But at least they are consistently defined in the app.**

12f. Glossary Build in www.ValPlan.net
ValPlan app can build user-tailored Glossaries in several ways:

a. User additions can be made to additional parameter libraries, such as defined Scales of Measure.

b. Local Definitions of Terms are automatically recognized and reused in a variety of ways: for eagle a '[Scale Parameter]' not by a user in Scale definition automatically gives rise to a field just below the Scale where we can define it. Then in all possible Scale points (Benchmarks, Targets, Constraints) it is automatically made available for reuse as a list of Scale Parameters (Like [Cities]) with their defined dimensions (like LA, NY, Oslo) ready for selection in a phase like 'Wish 60% [City = Oslo….]' .

c. Project Glossary: any term defined in project an be put into a project wide glossary.

d. Corporate Glossary: any project glossary term can be collected in a corporate wide glossary.

13. Source of Free Papers and Downloads on Planguage.
Continuously updated. Usually with a Twitter and Linkedin announcement.
http://concepts.gilb.com/file24

14. 'How Well Does the Agile Manifesto Align with Principles that Lead to Success in Product Development?' by Tom Gilb
and
• 'Why Agile Product Development Systematically Fails, and What to Do About It!' by Kai Gilb

and quite a few links to our other books and papers.
26 Feb 2018 in SyEN

https://www.ppi-int.com/wp-content/uploads/2018/02/SyEN_62.pdf

15. Design Logic Papers. (Gilb)

15A. "Ten Design Principles: Some implications for multidimensional quantification of design impacts on requirements", 2006

http://www.gilb.com/dl42

15B. Design Evaluation, Paper
"Design Evaluation: Estimating Multiple Critical Performance and Cost Impacts of Designs"
http://www.gilb.com/dl58

15C. The Logic of Design: Design Process Principles.
Tom Gilb, 2015, Paper.
http://www.gilb.com/dl857

16. Prioritization Writings
16A. Managing Priorities (paper)
http://www.gilb.com/DL60

16B. Choice and Using Planguage:
A wide variety of specification devices and analytical tools. (paper)
http://www.gilb.com/DL48

16C. Prioritization (about 60 pages of the VP book[5])

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

17. PLanguage (PL)

'17A. **Planguage: an 'engineering' language and process for real software and Systems engineering - not 'programming'**
http://www.gilb.com/dl831
Oct 18 2014 for
Book Chapter: originally intended for or "Software Engineering in the Systems Context" Lawson/ Jacobson eds., but replaced by another paper. Not published yet

17B. **Plicons: A Graphic Planning Language for Systems Engineering**
Copyright © 2006 by Tom Gilb. .
http://www.gilb.com/DL37

17C. **Planguage A Software and Systems Engineering Language, for Evaluating Methods, and Managing Projects for Zero Failure, and Maximum 'Value Efficiency'**
Keynote , Slides.
International Conference on Software Process and Product Measurement (Mensura)
http://concepts.gilb.com/dl918

18. Principles

18A http://www.gilb.com/dl352

INTRODUCTION: I made this collection primarily by cutting quotations from 'Competitive Engineering'. Not least I hope that those who currently enjoy quoting principles on Twitter, will have better access to these ideas. ⍰ I also hope to put a focus on the concept of useful principles. In my 1988 book, Principles of Software Engineering Management (I suspect one of the few books with 'principles' in the title, that actually has some principles in it?), http://www.amazon.com/Principles-Software-Engineering-Management-Gilb/dp/0201192462), someone once counted that I had about 120-140 specific principles. I hope to, one day (soon? Encourage me), publish that collection on the web. I can tell the reader that when writing CE, I did not 'peek' at the PoSEM principles, hoping to enhance my own creativity. Have fun! TomINTRODUCTION: I made this collection primarily by cutting quotations from 'Competitive Engineering'. Not least I hope that those who currently enjoy quoting principles on Twitter, will have better access to these ideas. ⍰ I also hope to put a focus on the concept of useful principles. In my 1988 book
n, Principles of Software Engineering Management (I suspect one of the few books with 'principles' in the title, that actually has some principles in it?), someone once counted that I had about 120-140 specific principles. I hope to, one day (soon? Encourage me), publish that collection on the web. I can tell the reader that when writing CE, I did not 'peek' at the PoSEM principles, hoping to enhance my own creativity. Have fun! Tom

http://www.gilb.com/DL352

18B. http://www.amazon.com/Principles-Software-Engineering-Management-Gilb/dp/0201192462

18C http://www.gilb.com/DL352

CE Principles Gilb and Others 8MB.
Extended Collection, Undergraduate basics Principles, Laws of Unreliability, Bill of Rights, Demarco and Gilb's Law of Measurability, Risk Principles, Clinical Risks Slides, GILB'S INTERPRETATION OF ERICSSONS CORPORATE QUALITY
POLICY, Decomposition Principles (from CE 10), 12 Tough Questions,

18D: The Previously cited [5] Value planning book is primarily based on a set of 100 Principles of Planning. About 7 pages of text for each principle.

18E. The book '**100 Practical Planning Principles**' is based on exactly the same set of principles as the value Planning book. https://www.gilb.com/store/4vRbzX6X, but is less than ⅕ the size.

19. **HP Evo experiences**
A. The Evolutionary Development Model for Software
by Elaine L. May and Barbara A. Zimmer
August 1996 Hewlett-Packard Journal
WWW.Gilb.com/dl67

B. Evolutionary Fusion: A Customer- Oriented Incremental Life Cycle for Fusion
by Todd A
www.gilb.com/dl35

August 1996 Hewlett-Packard Journal

C. RAPID AND FLEXIBLE PRODUCT DEVELOPMENT: AN ANALYSIS OF SOFTWARE PROJECTS AT HEWLETT PACKARD AND AGILENT (2001)
by
Sharma Upadhyayula

www.gilb.com/dl65

M.S., Computer Engineering University of South Carolina, 1991
And
Massachusetts Institute of Technology
January 2001

D. Best Practices for Evolutionary Software Development
by
Darren Bronson
http://www.gilb.com/dl825
Submitted for: Master of Business Administration
and Master of Science in Electrical Engineering and Computer Science
in conjunction with the Leaders for Manufacturing Program

At the Massachusetts Institute of Technology
June 1999
57 pages., 1999.

URI: http//hdl.handle.net/1721.1/80490

E. Note: these references focus on Evo part of Planguage. There is also a body of Literature for the HP Adoption of our **Inspection** method (alias **Spec QC**, in Planguage).
Grady, R. B. and Van Slack, T., “Key Lessons in Achieving Widespread Inspection Use”, IEEE Software, V. 11, N. 4, Month, 1994, pp. 46-57
http://dl.acm.org/citation.cfm?id=140207 (paid download)

20 . **Intel Planguage Experiences**
A. 2011 **Intel Report on SQC** (Gilb methods used here <- E Simmons)
‘The Impact of a Requirements Specification on Software Defects and Other Quality Indicators’ by john.terzakis (CONFERENCE TALK SLIDES)
http://selab.fbk.eu/re11_download/industry/Terzakis.pdf

B. **Intel Experience with Planguage and SQC 2011**
Erik Simmons, Intel, 2011, 21st -Century Requirements Engineering: A Pragmatic Guide to Best Practices, Erik Simmons PNSQC 2011 (Pacific Northwest Software Quality Conference)
http://www.uploads.pnsqc.org/2011/slides/Simmons_21st_Century_Requirements_slides.pdf

C. John Terzakis. 2013 Conference Paper
"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.
https://www.thinkmind.org/download.php?articleid=iccgi_2013_3_10_10012

21. **Billy Koen (Engineering heuristics (= Principles) as fundamental to engineering)**
21A: **DISCUSSION OF THE METHOD**

Conducting the Engineer's Approach to Problem Solving

Dr. Billy Vaughn Koen

Published by Oxford University Press, March, 2003

21B. His faculty Website
http://faculty.engr.utexas.edu/koen

21C: Video Lecture: **An Engineer's Quest for Universal Method.**
http://faculty.engr.utexas.edu/koen/etc-lecture (link tested 2019-6-10)

21D: “The use of engineering heuristics to cause the best change in a poorly understood situation within the available resources” (paper, Koen)

- Proceedings of the ASEE-IEEE Frontiers in Education. 14th Annual Conference, Philadelphia, PA. 3-5. October 1984. Pages 544–549. The paper also appeared in Engineering Education. December 1984. Pages 150–155. Also in Spring 1985 in The Bent of Tau Beta Pi. Pages 28–33. A full page extract is in Gilb (1988, Principles of Software Engineering Management). An extended and very interesting comment on the paper's ideas is in Koen (2003)
- I requested a URL from Koen 100619. I have a paper copy if necessary at home.

22. L Brodie PhD
Lindsey Brodie, PhD, 2015
Title: **"Impact Estimation: IT Priority Decisions"**
Middlesex University online publications.
http://eprints.mdx.ac.uk/18408/

23. Planguage Standards in addition to those in Competitive Engineering [2]

23A. **Evo Project Initiation Syllabus** 120519
'Evo Project Startup Standards: Day by Day'
http://concepts.gilb.com/dl946

23B. **Evo Standard**
ww.Gilb.com/dl487
2012 for DB, Non Confidential
See also [2] CE book for Evo Chapter standards.