Everyday Superpowers

by tom@Gilb.com

Version 020118, 080218 (ref edit)

There are a number of methods that I teach, which I have begun to headline to students as intellectual 'superpowers', when I lecture-to, or 'coach' people.

Intellectual Superpowers give you especially powerful abilities to analyze, communicate, present, and decide

Here is a list of the superpowers I will discuss in this paper.

1. Quantify value
2. Decompose solutions
3. Quantify and measure the quality of technical specs
4. Estimate solution attributes
5. Identify good solutions
6. Learn numerically what works
7. Prioritise fact-based solutions and requirements dynamically.
8. Quantify risks
9. Understand stakeholders
10. Communicate clearly

These superpowers are described in detail in my books and writings [1,2,3]. This paper is an introduction intended to stimulate further study to master the powers.

Here is one view of what a superpower is:

First, these are the **cultural** aspects of the superpowers:
- most people have not *learned* them yet
- most people do not do them *intuitively*
- many people will deny that it is *possible* to do them
- other people will argue they are too *difficult* to do
- most *university teachings* totally *ignore* these methods
- most *books*, talks, and papers do *not teach* these methods

In other words, they are rare: less than 1 out of 100 people can do them. If you can do them you are in the upper 1% of capability in this area, at least.

Here are the teaching and learning aspects of these superpowers:

- they are relatively quick to learn, within an hour or day each
- almost anybody in the professional area can learn to do them
- whole organizations can make them part of their culture
- they can be added one-by-one to the capabilities of any individual (yes, YOU)
- they can be added to any larger methods or processes, one by one
- it is easy to teach and coach them
- they can be taught 'on the fly', as you work and discuss or plan.
- the methods are described in detail in cheap, or free, sources [1, 2, 3]

Here are the expected results of using these superpowers:

- order-of-magnitude improvements in saving time and money
- scale free: the methods work at any scale of problem or project
- lean: the powers help deal with problems upstream, early and to prevent problems.
- agile: the superpower methods work in agile value delivery cycles, and support them.

So here is some detail:

I will keep to 'one page' per superpower. Read references for more detail.

1. Quantify value [4]

Values are all those things we (stakeholders) 'value'. We want i of them to i. We want *more* of them to be *satisfied*. Sometimes there is no limit to how much we dream that we want.

Values can be *practically anything* people can imagine. There is no finite, or limited, set of values.

Values are the primary reason we *make plans*, and *do projects*: we want to achieve improved levels of values, as soon as possible.

We are normally dealing with *more than one* value simultaneously. In fact the number of desired values for a set of stakeholders, for a 'project', normally exceeds 10, and values can seem to never end. So for this reason, we recommend that you focus on the 'top ten'; the Highest Priority Critical Stakeholder Values, at any one time.

And shove all other desires into a waiting area, to be considered *after* you have delivered or mastered production of the initial 'top 10'. [5]. I also recommend that this top 10 'requirements' or 'objectives' be drafted, and quantified, on the *first day of a project* [5, 6].

A key observation is that all 'values' are variable. We can always use terms like 'improved' in front of them ('improved security'). This is a signal that we *can* 'quantify' them. That means we can define a 'scale of measure' (like bits per second), and we can put a number of the 'level we desire' (like; '50 million bits/second by next January').

We have observed that there seem to be no exceptions to the rule that *all values can be quantified*. This is one key idea for many planning and decision-making applications, such as the other superpowers below, like Communicating Clearly.

It is common that people declare that *some* value ideas 'cannot be quantified', but this is easily disproved, and is actually a sign of their ignorance and 'arrogance'.

How do we quantify?

Method 1: Google it.

If you write the name of a value, followed by the word 'metrics', you will get many practical suggestions, from experienced experts, on ways to quantify the value. There are normally *many* quantification possibilities, several of which can be used in parallel. Try it.

Method 2: Look it up in a textbook.

You can also Google your way to a textbook, rather than just a paper or a webpages. For example my books give a great many systematic examples [1, 2, 3]. My chapter 'Scales of Measure [7] is a reasonable general collection of quality values such as adaptability, security and usability. These can be thought of as *general patterns,* and there is built-in scope for *tailoring* them any way you like or need. The tool needsandmeans.com has built-in scales of measure, initially borrowed from my CE book [7]. Studying, copying and tailoring previous scales of measure from other projects is quite useful too.

Method 3: Work it out, using 'domain understanding'

It will almost always work out, to ask a small team of stakeholders to craft a special scale of measure, and desired levels of value performance, and value level deadlines. Learning ideas of quantification such as the above 2 methods, or having an experienced coach on first try helps a lot.

2. Decompose solutions. [9]

Solutions are defined as the *means* by which we propose to achieve our value levels. Synonyms for 'solutions' are *strategies, architecture, design, means objectives.*

The solution problem, requiring 'value decomposition', is that some solutions will take far too much money and time to implement *fully* (years, and millions). We are interested in some faster-cheaper results, by means of partial solutions. This is a prerequisite for agile value delivery cycles, and for 'lean' (getting results early, and learning early, and *preventing* problems downstream).

Most people can decompose a solution into *some* parts, but the problem is that these parts *alone* will not deliver any value. Example, 'just supply the database, without any applications yet'.

So we have to constrain decomposition using some rules. The simplest practical teaching method, I use, which works quite well, goes like this:

1. Decompose to about 10 solution sub-components.
2. Make sure that each one can be implemented *before* any of the others: no *dependencies*.
3. Make sure each one, if implemented, can be expected to deliver some measurable, planned, value increment, to *at least 1* stakeholder. Rather than no-value-delivered at all.

This concept of independence (2 above), and value deliverability (3 above) is not immediately obvious to everybody. They will often in spite of my instructions, try to decompose into a *sequence of tasks* (Prepare Requirements, Do Architecture, Code the App, Test it. But their coach needs to spot this early, and ask the simple question "How much 'measurable value', do you expect, as a *result of implementing*, that suggested sub-solution?" And when the answer is 'none', then they need to think about what we are asking them to do (1, 2, 3 above), and try again.

This superpower enables you to get real measurable results early; which most people see as a good idea! This superpower is fundamental to the agile methods.

Designs

Design Ideas ->	*Technology Investment*	*Business Practices*	*People*	*Empowerment*	*Principles of IMA Management*	*Business Process Re-engineering*
Requirements	50%	10%	5%	5%	5%	60%
Availability 90% <-> 99.5% Up time	50%	5%	5–10%	0%	0%	200%
Usability 200 <-> 60 Requests by Users	50%	5–10%	5–10%	50%	0%	10%
Responsiveness 70% <-> ECP's on time	50%	10%	90%	25%	5%	50%
Productivity 3:1 Return on Investment	45% 50%	**R→ D Impacts**			100% 15%	53% 61%
Morale 72 <-> 60 per month on Sick Leave						
Data Integrity 88% <-> 97% Data Error %	42%	10%	25%	5%	70%	25%
Technology Adaptability 75% Adapt Technology	5%	30%	5%	60%	0%	60%
Requirement Adaptability ? <-> 2.6% Adapt to Change	80%	20%	60%	75%	20%	5%
Resource Adaptability 2.1M <-> ? Resource Change	10%	80%	5%	50%	50%	75%
Cost Reduction FADS <-> 30% Total Funding	50%	40%	10%	40%	50%	50%
Sum of Performance	*482%*	*280%*	*305%*	*390%*	*315%*	*649%*
Money % of total budget	15%	4%	3%	4%	6%	4%
Time % total work months/year	15%	15%	20%	10%	20%	18%
Sum of Costs	*30*	*19*	*23*	*14*	*26*	*22*
Performance to Cost Ratio	*16:1*	*14:7*	*13:3*	*27:9*	*12:1*	*29:5*

Source: Gilb, CE book, page 284. Persinscom US Army Personnel System.

An advanced tool for decomposition is the Impact Estimation Table (above) [1, 2, 3]. It decomposes the system by requirements (9+2) and Solutions [6]. About 2 orders of magnitude.
By searching for good impact (value, cost, or value for costs), like the 200% (2x Goal level), we can spot an opportunity. In this case for delivering an improvement in Availability, by *some* subset of Business Process Engineering. We found 2 opportunities, and implemented one, same day.[8] A 'Quick Win'.

3. Quantify and measure the quality of technical specs. [13]

We all understand the concepts of 'reviewing' a technical specification (like 'requirements', 'contract', 'architecture'). And we can all understand a concept of sign-off, or approval, of a specification. But very few (1% or less) know anything about 'measuring' the degree of 'goodness' of a technical specification. We have however practiced this extensively since about 1980 when IBM invented 'Inspection' for technical specifications [14]. Some major corporations, clients of ours, have practiced it pervasively like Boeing, Ericsson, Intel [15], IBM, NASA, and HP. But most organizations of today are totally ignorant of it. The CTO never heard of it. The universities do not teach it. Quite sad because this superpower will result in 10x, 100x and more reduction of upstream defects, with large corresponding reductions in being late, over budget, or failing to deliver enough quality.

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%***				

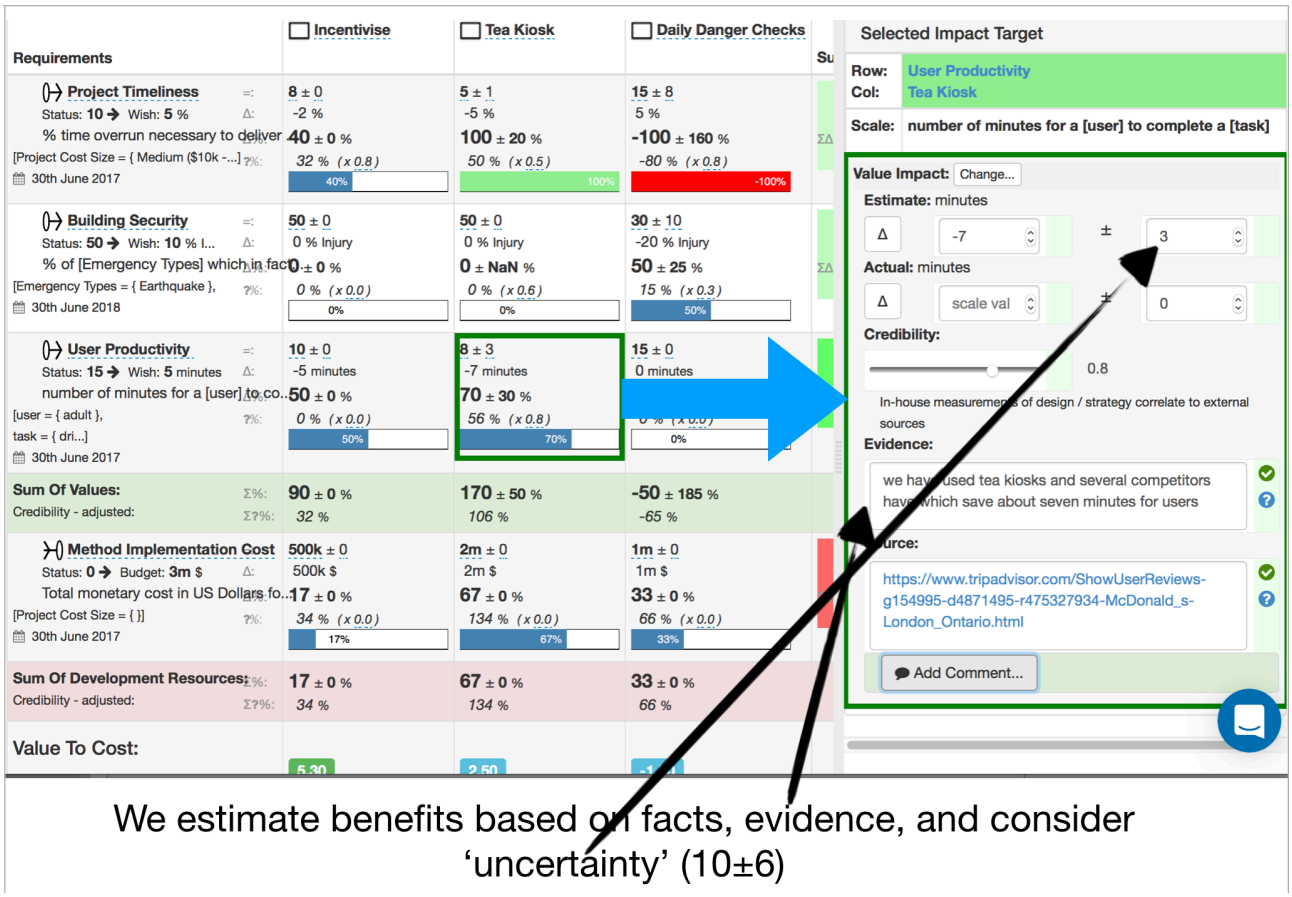
Terzakis' (Intel, [15]) shows a real example, above. He comments that it leads to 233% more delivered scope (engineering productivity). This is a result of *strictly demanding*, a *very high* requirement quality, before allowing anyone to use them for other purposes; before 'process exit'.

The technical specification measurement process, which we call 'Specification Quality Control' [1,2, 13, 15] or SQC, or Spec QC, simply asks 'checkers' to count Rule violations. The Rules are simply 'your current standards' for that specification type. Rule *violations* are called specification **defects**. The most 'generic' rules, applying to all tech specs, insist on unambiguous clarity.

In the Intel example above, the initial submission (for an SQC Review process), finds a density of 'only' 10 defects per 600-words (Page). Much better than the 100-300 defects per 300-word page which *you* have today, and do not measure, and do nothing about. But in this case Intel rejects the specification (no 'exit', defect level is initially 47x worse than the maximum allowed).

The result is that the requirements team is *motivated into learning*, to conform to the Intel Standards; which takes them 5 more rounds of review. But the result is 47x (10.06/0.22) better quality as we start serious use of the requirements, moving downstream, to architecture and testing. With any luck at all, this team has been motivated to learn to do more-rigorous specification *next* time, and will go straight to suitable levels. Doing it right the first time (lean).
Intel in this case is about 10 x 47 = 470 times better than you probably are: SUPERPOWER.

4. **Estimate solution attributes** [10 Sub-Ch 6.5, 16 (Strategies (=Solutions))]



We estimate benefits based on facts, evidence, and consider 'uncertainty' (10±6)

It is very fundamental to a logical system development process [17, Logic of Design] that we can estimate the multiple impacts of any solution proposal, on our critical requirements (values and costs). If we *cannot* do this, then we cannot *safely* decide what to do, and we run very high risks of failure (IT total or partial failure is 40% to 90%, Google 'IT project failure').

Estimations have their perils, but <u>not</u> estimating is suicidal.

Impact Estimations *do not*, and *cannot, predict* the results of solutions *accurately* (like ±5%) but they can hopefully give us 'order of magnitude': a normal engineering aspiration. And hopefully they will give us *insight* into the uncertainties, and into the 'more-certain solutions'. And hopefully they can lead to decisions to pilot-test solutions, *before* scaling up and committing. And hopefully estimations can give us a pretty good tool for agile *prioritization*: the things we *should do early*, in order to get *most value for resources*, with regard to *risks*.

The important idea is to '*know what you don't know*', and to '*know how uncertain you are*' about your knowledge. The worst situation is, *your* normal situation today, that you have not *ever* tried to estimate, *all the effects* of your solutions, on all of your *most-critical requirements*: i.e. on *your* success-and-failure value-and-cost levels. You are consequently **Powerless!**

Don't work harder, work smarter (my IBM manager's saying, 1962).

*The discipline in the Impact Estimation Table (IET), example above, is simple, but it gives the **superpower*** of understanding all proposals, solutions, options, architecture (i. e. all tech specs) etc. *before* costly committment.

You need to estimate the impact of each solution, on each planned value-and-cost requirement. In simple terms, '0%' means no impact. '100%' means we expect to reach our Goal on time. This can be difficult, but it is better than the alternative of project failure.

5. Identify good solutions:

How do solve a problem like Maria? (https://www.youtube.com/watch?v=M1HwVmY28Pk). Go on enjoy the song for a moment!

I define a '**good** solution' (or '**useful**') as one which has the capability of moving us *towards* our objectives, *within* our constraints. A '**better**' solution (more *economic*) moves us equally far towards our goals, using *less* resources, **or** (more *valuable*) 'moves us further towards our goals at the same costs'. A '**perfect**' solution moves us to *all* of our *critical* numeric goals, on *time*, within *budget*, at the end of the famous day, for real.

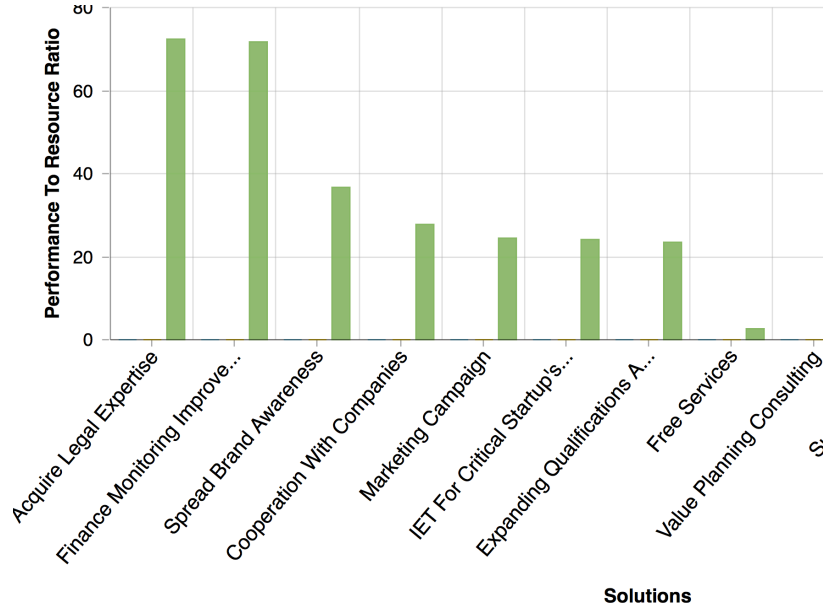
This means that:

1. we cannot identify good solutions without defining our 'critical requirements' numerically.
2. we cannot identify good solutions without knowing (facts, evidence, guarantees, experience) about the *degree of impacts* expected from a solution.
3. if even one single attribute of a solution (like a security level, or a maintenance cost) threatens, or risks, to take us below the 'minimum acceptable level' we specified (worst case), then the entire solution is invalid. The operation is NOT a success, then 'the patient dies'.

https://www.barrypopik.com/index.php/new_york_city/entry/the_operation_was_successful_but_the_patient_died

There are some interesting observations here:

1. the 'next new solution' you add to the solution mix, can threaten to destroy the success of *all previous* solutions in your total solution set.
2. nobody really knows how solutions will interact (sort of like cooking or chemistry, or engineering) in the short term and the long term: so you will ultimately have to 'suck it and see': and if it tastes sour, spit it out, fast. https://www.urbandictionary.com/define.php?term=Suck%20it%20and%20See
3. the only response to a failed solution that in fact fails you, is to rapidly and early discover it (measurement and learning) and change or replace it.
4. this means we have to implement the solutions sequentially, in an agile incremental mode; we cannot *finally* commit to *anything*, even if initially works fine. [Cleanroom, 18, 19]. Architecture needs to be decomposed, and implemented in small measured steps, to prove what happens as soon as possible, with as little failure risk as possible. Agile and lean.



The solutions for a Polish-Startup Export-plan (2017, training exercise, Warsaw) ranked by all performance values, for all costs (resources). This is derived from data on the Impact Estimation Ta-

ble. It gives a simple presentation of your probably best solution options. **Super-vision for complex problems.**

6. Learn *numerically* what 'works'.

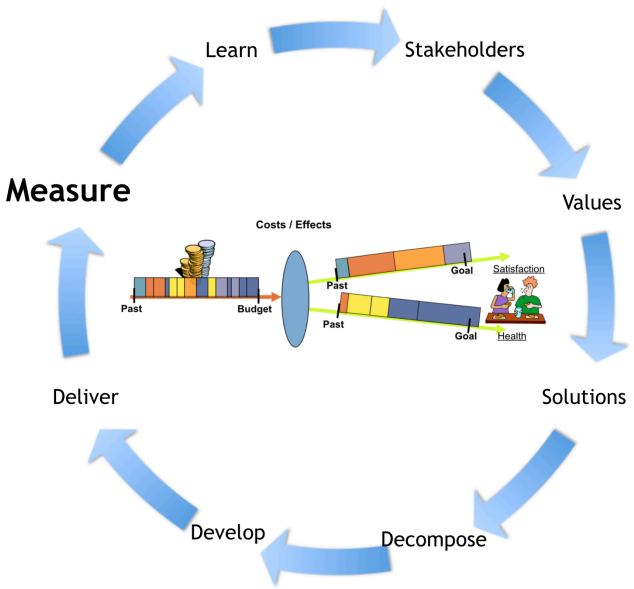
	A	B	C	D	E	F	G	BX	BY	BZ	CA
1											
2		Current Status	Improvements		Goals			Step9			
3								Recoding			
4								Estimated impact		Actual impact	
5		Units	Units	%	Past	Tolerable	Goal	Units	%	Units	%
6					Usability.Replacability (feature count)						
7		1,00	1,0	50,0	2	1	0				
8					Usability.Speed.NewFeaturesImpact (%)						
9		5,00	5,0	100,0	10	15	5				
10		10,00	10,0	66,0	20	15	5				
11		40,00	0,0	0,0	40	30	10				
12					Usability.Intuitiveness (%)						
13		0,00	0,0	0,0	0	60	80				
14					Usability.Productivity (minutes)						
15		20,00	45,0	112,5	65	35	25	20,00	50,00	38,00	95,00
20					Development resources						
21			101,0	91,8	0		110	4,00	3,64	4,00	3,64

Here is a real example, of learning what works (2003, Confirmit, [20]. A sub-team of 4 developers, decided to implement an architecture component called (Marketing Information) 'Recoding' (3/BY), because they estimated they would save 20 minutes (15/BX) of the planned 40 minutes saving, to their Goal level (15/E - 15/G = 40). Four days later (21/BX), (their agile measured result delivery cycle), Microsoft Usability Labs measured the effect, and the partial solution saved 38 minutes (15/BZ). They were so close to 100% of their goal (15/CA 95%) that they decided to make it 'over the hill' by means of one person putting in weekend overtime, and in fact ended up at 20 minutes, 12.5% more than the goal (15/D). Having reliable numeric feedback allowed them to make smart decisions, which I call 'Dynamic Design to Cost'. This is identical in principle to the IBM Cleanroom Method by Quinnan [18, 19].

With this '**superpower**' they *crushed* their competitors on the world market by delivering over 25 (in 3 mo.) incredible customer-visible quality improvements. And many more in 2nd qtr.

Observations:

1. most projects (yours, I assume) have **not** defined all their critical values (in this case 'Usability' which was 'market critical', worth 100x more business from Boeing and Microsoft alone), so they cannot relate 'value improvement measurement' to their 'planned objectives'. They do not know what to do with actual feedback from reality. They have no 'value budget' [20], so - what does the 'accounting' (numeric feedback after delivery) mean?
2. most projects have no process of *actually measuring* the result of a small solution-increment. So they cannot draw any 'learning' and corresponding 'action' from the solution delivery.



Our agile value delivery cycle.

Measuring actual results at each small decomposed solution delivery cycle, is necessary in order to keep your ship 'on course' towards success.

© Kai Gilb (Value cycle)

7. Prioritise fact-based solutions and requirements dynamically. [10, VP Book Chapter on Prioritization]

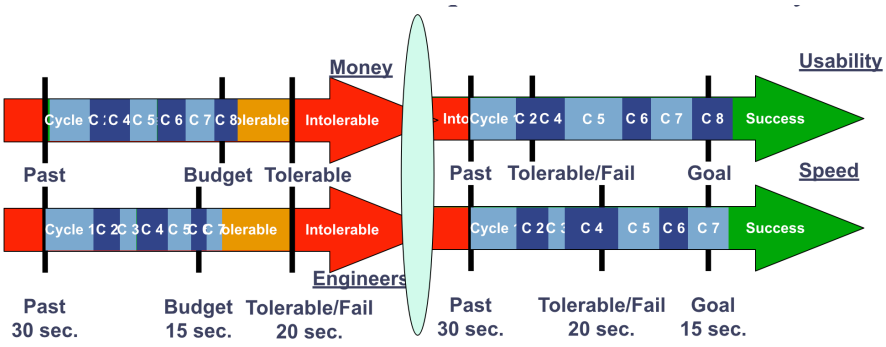
Smart prioritization of the 'sequence of system development actions' is a **superpower**. Most taught methods, for example Balanced Scorecards, and Quality Function Deployment [21] prioritize using subjective, anonymous, no-reason-given, static, up-front - priority 'weights', for example 1 to 5 stars, or 0-6 numbers. This is a widespread 'priority' method, and I argue that it is *not* suited for purpose, in complex system engineering [21, 10]. I am in fact astounded that such a bad method of prioritization has become so popular, and widespread, for so long, with almost no intellectual challenge to it.

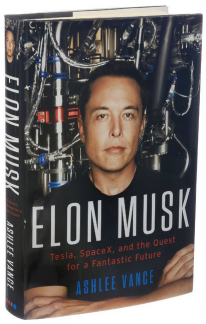
The desire for simplicity has outweighed the need for intelligent capability again. But the lack of challenge from academia, at least tells us that academia is *not* capable of fulfilling its role properly. One London professor explained this to me: the stupid fee-paying children of rich foreign parents would not be intellectually incapable of understanding anything more complex: and keeping their University 'business', is more important than doing the right thing! What a sad comment on humanity today. *Organizations* cannot afford to be misled by bad teachings. But can organizations do the right thing without dangerous oversimplification?

Simple observation of our own body prioritization, for survival and comfort, with respect to food, water, air temperature, sex, and human interaction, tell us clearly what the 'great architect of nature' has given us to prioritize. And it is not pre-birth fixed weights. It is dynamic calculation of the smartest **short term** actions to enhance comfort and survival.

We copy this behavior pattern in Planguage [1, 2, 3]. We set different levels of our multiple objectives for survival, for comfort, and for success. We then incrementally (short term) compute our priorities. *Survival is* highest priority (breath now, or suffocate), next priority - if no survival issues outstanding, next priority is *comfort* (can you turn up the thermostat in the car), and lastly *total satisfaction*, 'no more is necessary' ('I couldn't eat another bite now', of this incredible healthy veggie gourmet dining experience).

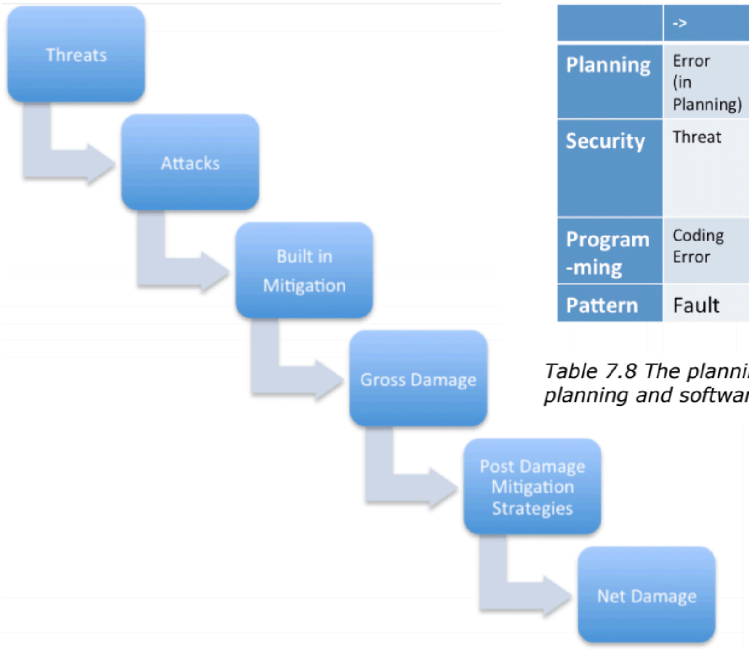
What has 'current priority', is *logically calculable*. Smart method! And smart people use it in their professional lives, not those silly Balanced Scorecard Weights, for unquantified requirements! Study Elon Musk, or any other smart businessperson or inventor. In Musks biography he points out that the Tesla production car gets about 20 improvements per week continuously. Half are new hardware in the production car, and half are over-the-air software improvements (like in smartphones). This is agile at its best. On my own, second, Tesla S, I note that the car is 'perfect' [22], exceeding my expectations, and getting better all the time!





When we iterate (agile) through solution delivery cycles (C1, C2, Cn) our priorities change; depending on the degree-of-satisfaction of our multiple critical values, and the concurrent depletion of our resources. The current cycle priority is logically 'computable'; both at design stages, and implementation, and operation stages. The red, yellow, and green signals in Column A 7-15, in the diagram from Confirmit (Superpower 6 above), is a real example of computed priorities at cycle 9 of 12 delivery cycles. The needsandmeans.com app has even more sophisticated automatic prioritization, based on costs and risks. (Diagram © Kai Gilb)

8. **Quantify risks** [11, Chapter in VP book on Risks]



	->	->	->	->	->	->	
Planning	Error (in Planning)	Specification Defect	Spec QC	System Defect	Malfunction or System Failure	Measure Value Delivery	Net Damage
Security	Threat	Attack	Mitigation (Built in)	System Defect	Damage (Gross)	Post Damage Mitigation (Recovery)	Net Damage
Program -ming	Coding Error	Code Bug	Inspection	Bugs Remain	Bug Encountered	Remove Bug	Net Bugs Remaining
Pattern	Fault	Fault	FIX	Fault	Fault	FIX	Fault

Table 7.8 The planning chain of areas of threats and mitigation, compared to 2 similar security planning and software reliability planning scenarios. © gilb.com 2014.

I have no understanding or sympathy about attempts to estimate, or quantify, the probability of a risk ('Threat' and 'Attack' above diagram) occurring. If it might happen at all, I need to consider dealing with it; or not. I do believe we can *estimate, with useful accuracy,* the *damage* caused by a successful attack, or series of them. We can estimate the costs for *preventing*, *detecting* and *mitigating* attacks. We might even be able to estimate the effects of anti-risk strategies.

I also believe that, as with all other values, we can test (using both test cases, and some reality) the ability of an incrementally-improved system, to deal with risks. We can get information on how well solutions work, and improve the design.

I am sure that all aspects of security, safety, and all other interesting risks, can best be approached by a systematic ***engineering*** approach, using numbers as a *basic tool*.

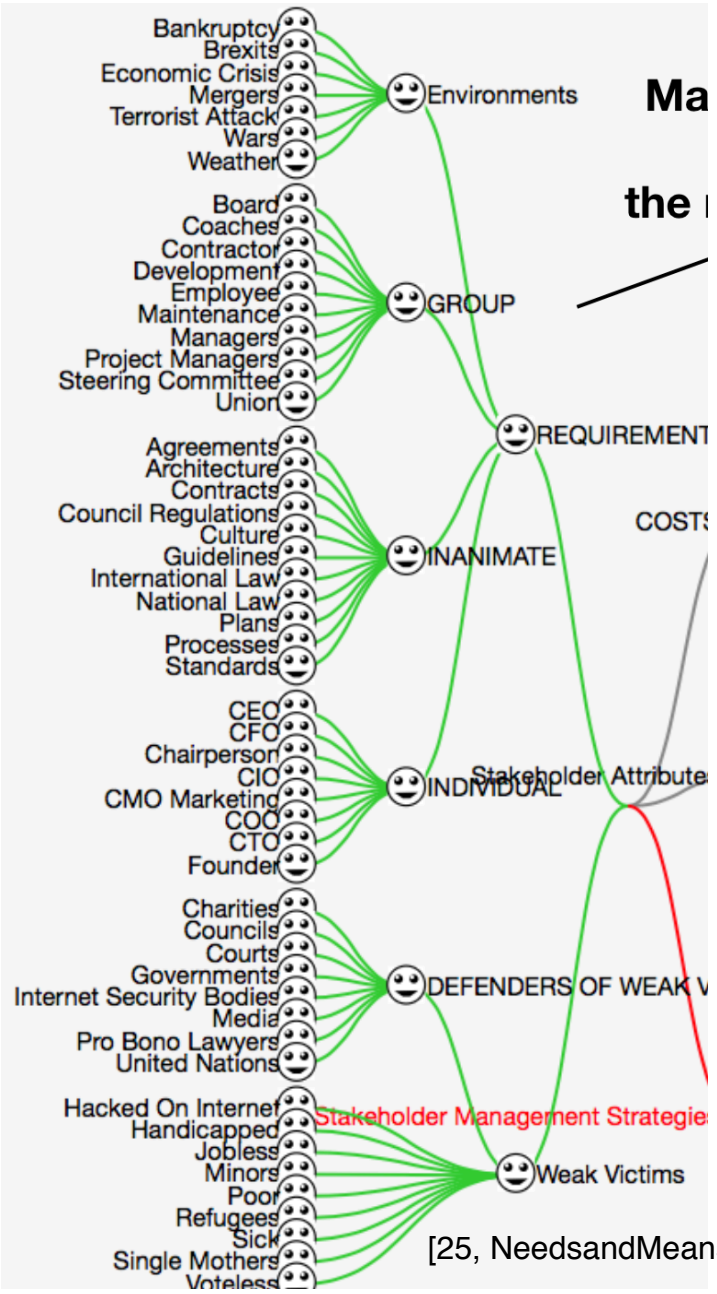
This can not be taken for granted. Years ago in Berlin, during the Cold War, a security course teacher from Washinton DC used a book he had written about security. I looked at it, and I could find a catalogue of hundreds of 'good ideas' for security, but absolutely no information for any of them about their quantified effectiveness for any attribute at all, nor any information about their costs of implementation and operation.

I told the Security Teacher, that his book did not given me any information that would allow me to select any security techniques. I asked him if he were ignorant of such information, or if he was hiding it intentionally? His answer was simply that 'nobody asks me for this data, I do know.'

So this tells us something about the state of ignorance we are in. We do not even ask for facts about subjects as critical as security ! Facts we could feed into an Impact Estimation Table, for example. We seem like our own worst enemy. Who is to blame? Our educational systems, and our management, right up to the top levels.

If we learn 'security engineering' we would indeed have a useful superpower. But who will 'bother'? Let me know if you are serious about security and safety and risk management.

9. **Understand stakeholders** [23, VP Chapter on Levels of Interest.]



[25, NeedsandMeans]

One superpower, well-known to systems engineering, like space and military projects, is **stakeholder** analysis.

But, there are too many cultures, outside of that, for example IT cultures and management cultures who have no relationship to 'stakeholders'.

Failure to identify and analyze critical stakeholders, is the first step in failure. We will not understand critical requirements, we will fail to design for these requirements and budget for them. We will fail.

The first step in a serious, large-scale, complex project is 'deep stakeholder analysis', and this needs to be a continuous process.

We need to invest in this discipline, stakeholder analysis, as a very 'Lean' approach to dealing with potential problems at the earliest possible stage.

10. **Communicate clearly** [12]

When all involved parties really understand exactly the same thing, as the original writer intended: then we have clear communication. And that perfect shared understanding is the pre-requisite for teamwork.

All the above superpowers will be tools for communicating more clearly.

Here are some principles of clear communication and Clear Thinking

1. You have to have a *clear set of objectives and constraints*, to evaluate proposed solutions or strategies against.

2. You have to have a r*easonable set of facts* about the *benefits and costs* of any proposed idea, so that you can relate it to you current outstanding requirements.

3. You have to have *some notion of the risks* associated with the idea, so that you can understand and take account of the worst possible case.

4. You have to have some ideas about how to *test the ideas gradually, early and on a small scale* before committing to full scale implementation.

5. If there are more than very few factors involved (2 to 4) then you are going to have to use a *written model of the objectives, constraints, costs, benefits, and risks.*

6. If you want to check your thinking with anyone else, then you will need a written model to *safely and completely share your understanding* with anyone else.

7. You will need to make a clear distinction between necessities (constraints) and desirables (targets).

8. You will need to *state all assumptions clearly, in writing*, and to challenge them, or ask 'what if they are not true?'

9. You will want to have a *backup plan*, contingencies, for the *worst case* scenarios – failure to fund, failure for benefits to materialize, unexpected risk elements, political problems.

10. Assume that information from other people is *unreliable, slanted, incomplete, risky – and needs checking*.

REFERENCES:

1. Value Planning (2017): leanpub.com/ValuePlanning, or 50% https://goo.gl/MB6kaR Code: CONNECT
2. Competitive Engineering (2005): Get a free e-copy of 'Competitive Engineering' book. https://www.gilb.-com/p/competitive-engineering
3. gilb.com downloads website, http://concepts.gilb.com/file24
4. TedX Talk: Tom Gilb, 'Quantify the Unquantifiable' https://www.youtube.com/watch?v=kOfK6rSLVTA
5. Gilb's Mythodology: Top Ten 'The Top 10 Critical Requirements are the Most Agile Way to Run Agile' Projects http://www.gilb.com/dl797
6. Gilb: An Agile Project Startup Week www.gilb.com/dl568
7. Gilb, Competitive Engineering, Chapter 5 Scales of Measure.https://www.dropbox.com/s/z031wn8s6aduvze/
8. in CE book [2] and See Persinscom Case: US DoD Army Personnel System. "111111 Unity Method of Decomposition into weekly increments of value delivery". (10 min. talk slides) http://www.gilb.com/DL451
9. See the Chapter 5 Decomposition chapter in Value Planning [1] or https://www.dropbox.com/sh/dc7v636m7w7vvgx/AABfMAW_FnJny23XZKQZQk-F4a?dl=0
10. Ch. 6 Prioritization VP book [1] ,https://www.dropbox.-com/sh/34llx1a7ckyagxl/AAA0pDzSxN5WmoP9l-OKR0Mpca?dl=0
11. Ch. 7, Risk Management in VP [1], https://www.drop-box.com/sh/fxvtya6gyvgwkfa/AAA5-vr-LUt_z0h9EYt1ql3Uma?dl=0
12. Ch. 9 in VP book [1], Communication, https://www.drop-box.com/sh/fxvtya6gyvgwkfa/AAA5-vr-LUt_z0h9EYt1ql3Uma?dl=0

13. Ch. 10 in VP book [1] 'Quality Management. https://www.dropbox.com/sh/vjwybhqfxrvctk7/AAAdab-ECBSo5x-tSOI85R-1da?dl=0
14. Gilb & Graham, Software Inspection, 1993
15. J. Terzakis, (Intel) "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
16. Ch. 2 Strategies in VP [1], https://www.dropbox.com/sh/xab857l9ksfs7w0/AACKonxV1x_LI5TW62FICMM-Pa?dl=0
17. "The Logic of Design: Design Process Principles". Tom Gilb, 2015, Paper. http://www.gilb.com/dl857
18. Cleanroom, Quinnan, in VP, case 2.5 [8], QUINNAN AND MILLS CLEANROOM http://www.gilb.com/dl821. **Cleanroom**
19. A. Mills, H. 1980. **The management of software engineering: part 1: principles of software engineering.** *IBM Systems Jour- nal* 19, issue 4 (Dec.): 414-420.
Direct Copy http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&con- text=utk_harlan
Library header
http://trace.tennessee.edu/utk_harlan/5/
B. Mills, Harlan D.; Dyer, M.; and Linger, R. C., **"Clean-room Soft- ware Engineering"** (1987). The Harlan D. Mills Collection. http:// trace.tennessee.edu/utk_harlan/18
20. Value Driven Project Management 17.5MB slides 2008 '152'. Includes Confirmit Case (slide 70-93). http://www.gilb.com/dl152
21. T. Gilb and L.Brodie, "How problems with Quality Function Deployment's(QFD's) House of Quality (HoQ) can be addressed by ap-

plying some concepts of Impact Estimation (IE)" http://www.gilb.com/DL119

22. Elon Musk, the bio by Ashlee Vance, 2015. The only fault in my Tesla S from Oct 2016 to December 2016 was that the remote tire pressure sensors did not work properly with non-standard winter tires. This was fixed for free by Tesla. That record is not an accident, it is a result of iterative intelligent prioritization weekly forever.
23. Value Planning, Chapter 3, Levels of Interest. About stakeholders.
24. https://www.dropbox.com/sh/xbzn5s8imf9vla0/AAB8h-OFvQm-J_w3wNhrDxa9_a?dl=0
25. Needs and Means Planning tool.
26. Slides

10 Consultant Superpowers
http://concepts.gilb.com/dl927

70 slides 20 MB
Feb 7 2018 at Ciber Oslo