Making Metrics More Practical in Systems Engineering: Fundamental Principles for Failure and for Success

Tom Gilb
Result Planning Limited
Tom.Gilb@INCOSE.org

Abstract.

Quantification is the essence of real engineering. Engineering is good at quantifying many traditional aspects of systems. But there are weak points. For example in quantification of emerging 'soft' aspects of systems like usability and security, and within the emerging sub-disciplines of software and data. We need to use the quantification tool in all critical aspects of of our systems, not just in traditional sectors. This paper will explore the extension and strengthening for the metrics discipline in the weakest areas of systems engineering.

Failure Prone Principles

Here are a set of principles, as a framework for discussing poor practice, and its avoidance.

1. ***If you measure what is easy rather than right, you'll lose the fight.***
2. ***If you measure too late, you deserve your fate.***
3. ***If you measure too few, then the ones you left out, will lack any clout. If you measure too many, you will also lose out.***
4. ***If the metric level is too low, your users are in for a sorry blow.***
5. ***Know the role of your metric, or it can roll over on your project.***
6. ***If you fail to quantify a critical variable, it will fail to be what you need.***
7. ***Do not trust managers to define the most critical metrics, help them out***
8. ***Some metrics support other metrics. You'd better know which is the star, and which is the supporting role.***
9. ***Metrics don't add up, but you need to understand the set of them.***
10. ***Metrics are a generally good tool, until they are used carelessly, or to manipulate people.***

These will be discussed in more detail below.

Success Principles

Here is another set of principles, less warning of bad practices, and more proscriptive of good practices.

1. Develop requirements metrics top down from critical management objectives.

2. Connect metrics with metrics.

3. Develop metrics with early rapid numeric and non-numeric feedback.

4. Use metrics to describe metrics, for credibility, and uncertainty

5. Use metrics to describe solutions, designs, and architecture.

6. Use multiple metrics to compare alternatives.

7. Measure critical variables, but with sufficient qualities and lowest costs.

8. Use metrics to review specifications.

9. Use metrics to prioritize, and determine priorities.

10. Use metrics to create commonly understood, and really agreed requirement or objectives.

Detailed Warning Principles

1. ***If you measure what is easy rather than right, you'll lose the fight.***

The drunk knew he'd lost his watch down the street in a dark corner,

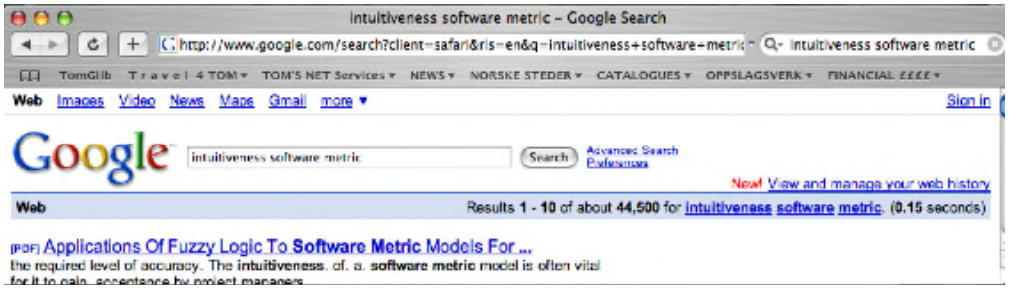
But it was tempting to look for it under the lamp post

Determine what is most critical to control,

and then find a way to quantify it - there is always a useful way

then find ways to measure that quantity

There are always useful ways



If you can't imagine the ways to quantify or measure something, the internet can.

Figure 1: an example of looking for metrics for 'soft' ideas, on the internet.

"In physical science the first essential step in the direction of learning any subject is to find principles of numerical reckoning and practicable methods for measuring some quality connected with it.
I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it;
but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind;
it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of Science, whatever the matter may be."
Lord Kelvin, 1893

Quotation 1: my own guiding light sine the 1960's – a word from the wise.

2. ***If you measure too late, you deserve your fate.***

You need to measure early, in order to discover -

- what to measure, what the requirements really are
- what measures are useful
- what is worth measuring
- what the actual numeric levels of requirements should be

Measuring at the end of a project,

- is just too late to learn in time
- to convince people that they have a solvable problem
- in time to solve it

	A	B	C	D	E	F	G	BX	BY	BZ	CA
1											
2		Current Status	Improvements		Goals			Step9 Recoding			
3								Estimated impact		Actual impact	
4											
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	0	15	5				
10		10,00	10,0	200,0	0	15	5				
11		0,00	0,0	0,0	0	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

Figure 2: a real example of measuring early in the development of a commercial product ('Confirmit'). The actual impact (38 minutes) of a design ("Recoding") implemented in the 9th week of evolutionary product development was measured by a pilot customer (Microsoft usability Labs, Redmond) at the end of the time-boxed week cycle. This gave the feedback that the design was better than expected (Estimated Impact = 20 minutes). It led to a decision to put in some overtime and squeeze a little bit more out of the design before the next cycle started. This led to saving 45 minutes (112.5% improvement). It also freed up the remaing 3 week cycles before product release to focus on weaker areas (like line 7,11, 13 at less than 100% of target to date).

3. If you measure too few, then the ones you left out, will lack any clout. If you measure too many, you will also lose out.

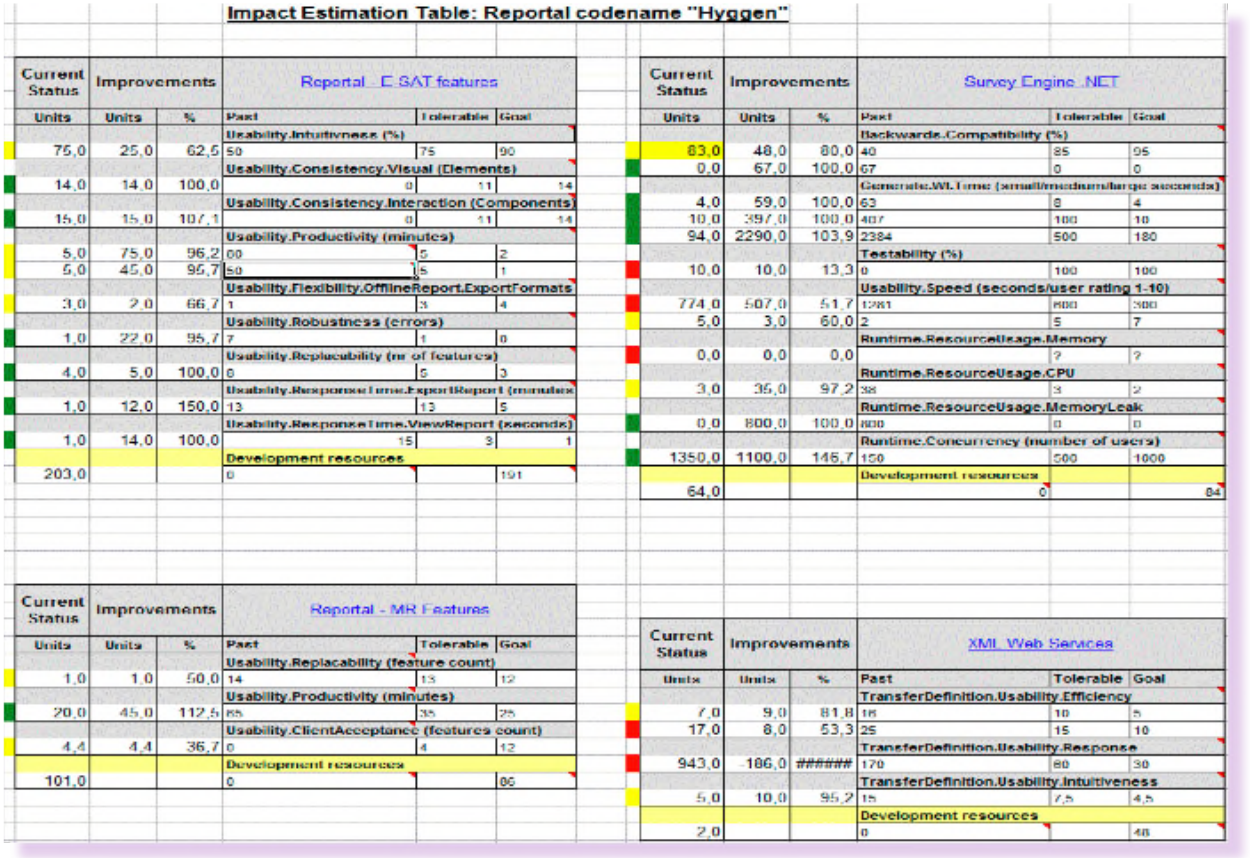
Limit yourself, at any one level of consideration, to the maximum 'top ten' most critical requirement measures.

When you have mastered all of them, you might have resources left to turn to the next priority requirement.

You cannot afford to distract your attention from the top few highest priorities

Mastering 10 critical variables, at demanding levels, is a magnificent technical management deed.

You will be forgiven for failing on the 11th, for the moment - it is next on your hit list anyway.



Impact Estimation Table: Reportal codename "Hyggen"

Current Status	Improvements		Reportal - E-SAT features		
Units	Units	%	Past	Tolerable	Goal
			Usability.Intuitivness (%)		
75,0	25,0	62,5	50	75	90
			Usability.Consistency.Visual (Elements)		
14,0	14,0	100,0	0	11	14
			Usability.Consistency.Interaction (Components)		
15,0	15,0	107,1	0	11	14
			Usability.Productivity (minutes)		
5,0	75,0	96,2	80	5	2
5,0	45,0	95,7	50	5	1
			Usability.Flexibility.OfflineReport.ExportFormats		
3,0	2,0	66,7	1	3	4
			Usability.Robustness (errors)		
1,0	22,0	95,7	7	1	0
			Usability.Replacability (nr of features)		
4,0	5,0	100,0	0	5	9
			Usability.ResponseTime.ExportReport (minutes)		
1,0	12,0	150,0	13	13	5
			Usability.ResponseTime.ViewReport (seconds)		
1,0	14,0	100,0	15	3	1
			Development resources		
203,0			0		191

Current Status	Improvements		Survey Engine .NET		
Units	Units	%	Past	Tolerable	Goal
			Backwards.Compatibility (%)		
83,0	48,0	80,0	40	85	95
0,0	67,0	100,0	67	0	0
			Generate.WLTime (small/medium/large seconds)		
4,0	59,0	100,0	63	8	4
10,0	397,0	100,0	407	100	10
94,0	2290,0	103,9	2384	500	180
			Testability (%)		
10,0	10,0	13,3	0	100	100
			Usability.Speed (seconds/user rating 1-10)		
774,0	507,0	51,7	1281	800	300
5,0	3,0	60,0	2	5	7
			Runtime.ResourceUsage.Memory		
0,0	0,0	0,0		?	?
			Runtime.ResourceUsage.CPU		
3,0	35,0	97,2	38	3	2
			Runtime.ResourceUsage.MemoryLeak		
0,0	800,0	100,0	800	0	0
			Runtime.Concurrency (number of users)		
1350,0	1100,0	146,7	100	500	1000
			Development resources		
64,0			0		84

Current Status	Improvements		Reportal - MR Features		
Units	Units	%	Past	Tolerable	Goal
			Usability.Replacability (feature count)		
1,0	1,0	50,0	14	13	12
			Usability.Productivity (minutes)		
20,0	45,0	112,5	65	35	25
			Usability.ClientAcceptance (features count)		
4,4	4,4	36,7	0	4	12
			Development resources		
101,0			0		86

Current Status	Improvements		XML Web Services		
Units	Units	%	Past	Tolerable	Goal
			TransferDefinition.Usability.Efficiency		
7,0	9,0	81,8	16	10	5
17,0	8,0	53,3	25	15	10
			TransferDefinition.Usability.Response		
943,0	-186,0	######	170	80	30
			TransferDefinition.Usability.Intuitiveness		
5,0	10,0	95,2	15	7,5	4,5
			Development resources		
2,0			0		46

Figure 3: an example of the top few product requirements. They were all quantified, and all measured as they were delivered in weekly increments. This is a snapshot of the 9th week. Look at the % improvements – it tells cumulative, independently measured progress towards the target for the quarterly product release. Each quadrant is a top ten requirement (or less) allocated to a parallel working development team, for the same product, but for different semi-independent components of it.

4. If the metric level is too low, your users are in for a sorry blow.

What is 'too low' a requirement level?

There are several simultaneous variations to consider:

- too low in relation to a future competitor level (uncompetitive)
- too low in relation to our current levels (worse product or service)
- too low in relation to constraints
- too low at a particular time
- too low in a particular area
- too low under specific conditions or events

In Planguage [Gilb 2005] we specify a number of benchmark relationships, so that the numeric requirements can be seen in the light of these relationships.

For example:

Repair:
Ambition: Improve the speed of repair of faults substantially, under given conditions.
Scale: Hours to repair or replace, from fault occurrence to when customer can use faultlessly, where they intended.
Meter [Product Acceptance]: A formal Field Test with at least 20 representative cases, [Field Audit]: Unannounced Field Test at random.
========== **Benchmarks** ==========
Past [Product
Phone XYZ, Home Market, Qualified Dealer Shop]:
{0.1 hours at Qualified Dealer Shop
0.9 hours for the Customer to
transit to/from Qualified Dealer Shop}.

Record [Competitor Product XX]: 0.5 hours average. !!Because they drive a spare to the customer office.'"

Trend [USA Market, Large Corporate Users]: 0.3 hours. !!As on-site spares for large customers.'"
========== Targets ==========
Goal [Next New Product Release, Urban Areas, Personal Users]: 0.8 hours in total, [Next New Product Release, USA Market, Large Corporate Users]: 0.2 hours <- Marketing Requirement, February 3 This Year.
========== Constraints ==========
Fail [Next New Product Release, Large Corporate Users]: 0.5 hours or worse on average <- Marketing Requirement, February 3 This Year.
Survival [Next New Product Release, All Markets]: 1.0 hours <- TG

Example 1: In 'Planguage' we strongly encourage collection and specification of benchmark levels of performance, so that the engineer is conscious of these facts, in relation to requirement levels (Targets, Constraints) – and is not too low or too high. Source [Gilb, 2005, pages 114-115]

5. Know the role of your metric, or it can roll over on your project.

A metric lives in a system environment

Spaces

Geographical, Market Segment, Task Type, …..

Time

Deadlines

Intervals ('office hours', 'weekends')

Obsolete times, irrelevant times, …..

Concurrent events and conditions

Contracts signed, laws in force, achievements succeeded, …..

We need to carefully define that environment - the metric 'role' in the 'play

In Planguage we have a large number of devices that encourage engineers to specify the relationship of anything (requirement, design, plan) to any other part of a total specification, and outside environment.

One simple example, is the way we specify requirements with '[Qualifiers]' – which are a list of conditions that must all be true for the specification to be effective or 'true':

Goal [Deadline = Release 9.0, Market = {China, HR Departments}, Condition = Merger with China Hotline OK, User = HR Employee, Tasks = Critical Tasks] 60%

Another Planguage tactic for specifying relationships are specification parameters such as:

- 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!
- And 'Qualifiers, like
- Goal [UK, Teens, 2009] 35%

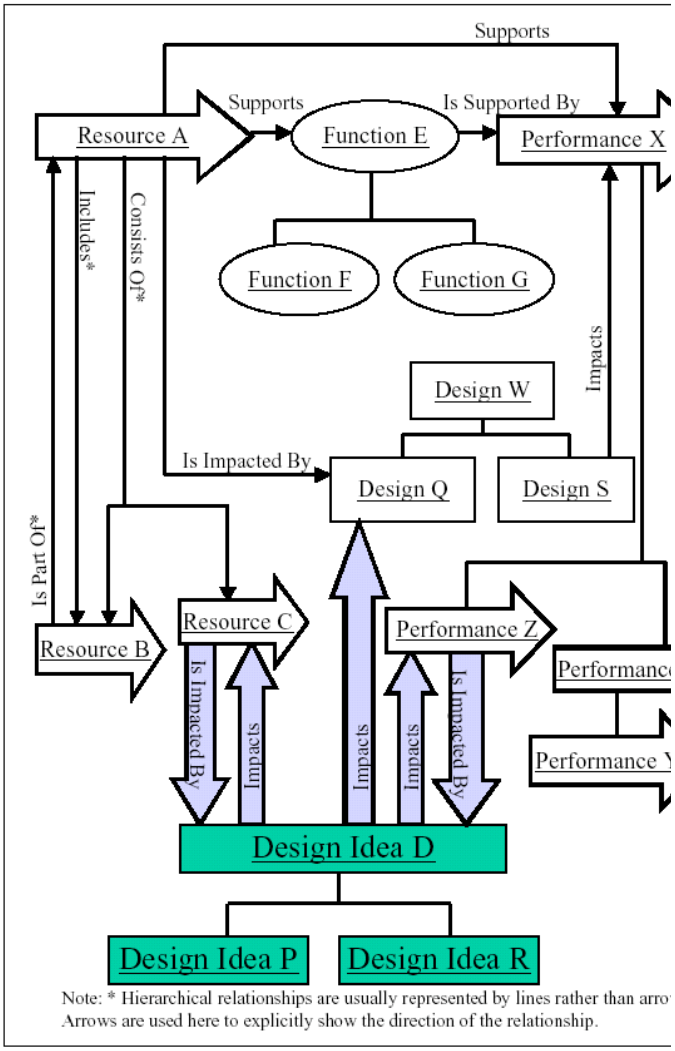


Figure 4: Specification options in Planguage to help us understand relationships.

6. ***If you fail to <u>quantify</u> a critical variable, it will fail to be what you need.***

Developers will naturally prioritize quantified target *requirements* that they believe they will be judged on delivering

And they will prioritize delivering to quantified *constraints* (deadline, budget)

So we need to have a notion of being 'complete' for the quantified critical requirements:

We cannot have some quantified and others equally important in un-quantified formats like

"Very User-Friendly", "Highly Secure", "Extremely Adaptable"

> *"All critical factors or objectives*
> *(quality, benefit, resource)*
> *for any activity*
> *(planning, engineering, management)*
> *shall be expressed clearly, measurably,*
> *testably and unambiguously*
> *at all stages of consideration, presentation,*
> *evaluation, construction and validation. "*

Quotation 2: A summary of the Ericsson Corp. Quality Policy position on quantification.

7. ***Do not trust managers to define the most critical metrics, help them out.***

Managers have no training or culture in developing quantified and clear metrics for their most critical qualitative ('soft') objectives.

They love to use a series of popular words, because that is their culture today.

They consistently call the strategies (means to ends) the 'Objectives'.

(get CMMI Level 3, rather than increase productivity by 20%)

If you guide them into quantifying their wordy objectives,

Some of them will love it and learn it. The CEO, COO, and CFO types will at least.

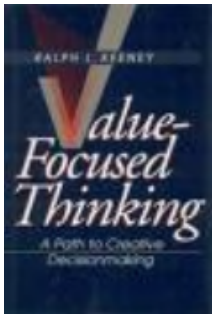
Some of them would rather lose their jobs (the marketing types especially).

Business objective	Measure	Goal (200X)	Stretch goal ('0X)	Volume	Value	Profit	Cash
Time to market	Normal project time from GT to GT5	<9 mo.	<6 mo.	X		X	X
Mid-range	Min BoM for The Corp phone	<$90	<$30	X		X	X
Platformisation Technology	# of Technology 66 Lic. shipping > 3M/yr	4	6	X		X	X
Interface	Interface units	>11M	>13M	X		X	X
Operator preference	Top-3 operators issue RFQ spec The Corp	1	2	X		X	X
Productivity						X	X
Get Torden	Lyn goes for Technology 66 in Sep-04	Yes		X		X	X
Fragmentation	Share of components modified	<10%	<5%		X	X	X
Commoditisation	Switching cost for a UI to another System	>1yr	>2yrs		X	X	X
Duplication	The Corp share of 'in scope' code in best-selling device	>90%	>95%		X	X	X
Competitiveness	Major feature comparison with MX	Same	Better	X		X	X
User experience	Key use cases superior vs. competition	5	10	X	X	X	X
Downstream cost saving	Project ROI for Licensees	>33%	>66%	X	X	X	X
Platformisation IFace	Number of shipping Lic.	33	55	X		X	X
Japan	Share of of XXXX sales	>50%	>60%	X		X	X
	Numbers are intentionally changed from real ones						

Example 2: One CEO client ("The Corp") of ours needed £50 million to improve his engineering organization for telecoms sub-components. He 'got the religion' about quantified results (as opposed to paying management bonuses for strategies delivered, without results) and asked his chief planner to quantify his 'Business Objectives' (quite technical most) to present to the money guys. It worked. (Corrupted for anonymity, and used after asking client permission)

8. Some metrics support <u>other</u> metrics. You'd better know which is the star, and which is the supporting role.

Ralph Keeney's Levels ('Value-Focused Thinking')



Keeney's Levels are all relative to your level in the organization

- Fundamental Objectives (Your boss level of concern)
- Strategic Objectives (you, your responsibility and concern)

- Means Objectives (your staff, and support, objectives that will contribute to meeting your own objectives)

This relative identification of the level of concern helps in understanding *responsibility* for objectives and requirements. You need to keep focus on your *own* strategic objectives, and communicate about those above, and those below, with appropriate people. Don't take it all on by yourself.

9. Metrics don't add up, but you need to understand the set of them.

The varied top ten objectives metrics cannot be directly 'added' to each other, to get a sum of improvements.

But the % of progress towards the 10 different Goal levels can be added and averaged to get some idea of progress to date.

We use an Impact Estimation table [Gilb 2005] to do this, and use the concept of % of goal achievement to have a unit of measure we can 'add up' for different objectives.

Figure 3 gives one example. See the column "Improvements, %" to get some idea. You could add and average the % impact on goals achieved on this 9th week of 12 before release (75 of time to deadline) to get an idea of whether the projects are on track (75% or more improvement) or not.

10. Metrics are a generally good tool, until they are used carelessly, or to manipulate people.

So we need

- sound best practice standards
- training
- management leadership
- quality control
- a constant learning process

The ideas and practices exist [All References], but the sound metrics culture and motivation to apply it is not there now, most places. We would prefer, it seems, to let our projects fail, than to get disciplined enough to avoid failure. We get paid anyway – failure or not!

Detailed Success Principles

Here is another set of principles, less warning of bad practices, and more proscriptive of good practices.

1. Develop requirements metrics top down from critical management objectives.

The most critical requirements in any project, are

The critical few improvements that the project sponsors are hoping for

They are 'always' quantifiable!

All other 'requirements' are in reality supporting requirements for the top ones.

At the 'top systems level' there are some 'stakeholder values' - like save time, sell more. These top level stakeholder values can always be expressed quantifiably, for clarity and mutual agreement of common objectives.

Software products can have performance/quality requirements to directly support delivery of these top level stakeholder values

Example of software product quality requirement, to support a top level requirement like 'save user time and cost':

Increase Usability (defined by some Scale, like 'Scale: Learning time for defined Tasks by defined Employee with defined Experience') by 50% (compared to present level), by next release

2. Connect metrics with metrics.

There are many types (Management Objectives, Product Quality Requirements, Project Effort Budgets)and levels (Survival, Fail, Goal, Stretch, Wish) of metrics

You should make their relationships and connections clear and documented.

Figure 4, above, lists some of the many Planguage tools for doing this in practice. Example 1, above, gives an example of how the levels are clearly related to a single requirement and scale of measure.

In the few instances where we see a metric used in requirement specification, for example:

From the H project ($100 million wasted with no delivery for 8 years by end 2006) Top Level Objectives:

Here on a single page, for a single major objective (Robustness) were the following 4 'requirements'. None of these had any more detail before a long list of technical design to achieve them was listed ('Built-in testability, Tool simulators').

"Minimal down-time"

"Built-in testability"

These two were without *any* attempt to quantify, as were *all* other top level requirements, totally un-quantified. Unintelligible – 'sales' talk maybe, but not engineering talk.

"Mean time between forced restarts > 14 days"

"Restore system state < 10 minutes"

These two had at least a number attached to them. And that gives us some idea of the order of magnitude we are in need of.

But, here is the problem, see if you can answer any of these questions.

1. is this level related to any contractual service levels with any customers, or *will* they be?

2. how much better or worse are these levels than *current* experience? 'Past' levels

3. are these levels just *wishful* thinking (Wish levels) or have any remotely similar systems *ever* achieved these levels? (Record Level =?)

4. Do these levels have to be achieved by any particular *deadline*? 'Goal [Release 10.0] 14 days.'

5. What is the *payoff* from reaching these goals for us and other stakeholders in the long term?

6. Are these goals to be achieved *irrespective* of costs? Is $500 million acceptable?

7. What are the *constraints* (worst acceptable levels) on these goals (Fail level, Survival Level, see Example 1), assuming these numbers *are* target(Goal, Stretch) levels (not constraints – who knows?).

8. Do these levels apply to any particular *subset* of the application, and particular *use* situation, scale of usage, key customers, or do they really apply to 'everyone', 'always'?

9. Which higher-level objectives would the achievement of these numbers *support*?

10. is there any lower levels of performance that might be *useful* improvements in *key areas* of application, with key projects and customers, in the shorter term, before these levels can be achieved. (Keep in mind nothing, in this real project, was achieved at all for the next 8 years!!).

I could easily add 50 more questions of the nature. But you get the idea.

You can imagine that the answer to these questions is important to know, in order to make *design* and *project planning* decisions, and in order to *prioritize* these in relation to other objectives. But the information is not specified *anywhere*. It could *all* have been made available (example using Planguage, or simple text) together with the 'basic' requirement.

There was absolutely no engineering culture, in this large engineering organization, to even produce a single page of text for these requirements, which in this example were 1/8 of a $100 million project spend! Maybe the roughly $12 million (1/8 of $100 million) spent on this would

have been more effectively spent if the metrics were backed up by more study and more information. I think so. Unfortunately – this organization is typical, rather than exceptional.

3. Develop metrics with early rapid numeric and non-numeric feedback.

You will, hopefully – if you focus right - be trying to get to a critical few numeric long-term goal levels - of performance/quality.

We believe the smartest way to the long term is to try to move towards them in *early, frequent, small 'weekly' steps*. (Real example Figure 2)

The frequent-cycle metrics are initially estimated – to decide what pays off to do;

then measured at cycle end, to confirm or deny the estimate-hypothesis;

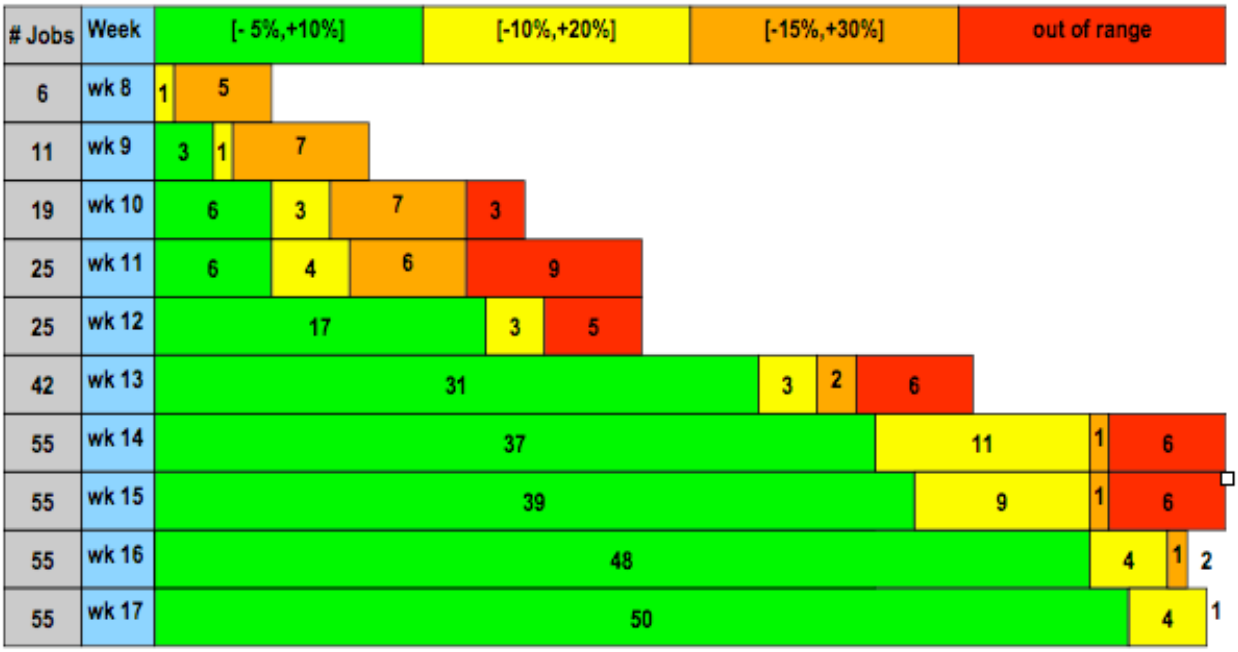
then evaluated against estimates, to learn from the degree of deviation from expectation.

This gets real results for stakeholders – rapidly. That builds trust, confident, interest.

This makes sure your entire development process works – if you find that the root cause of failure to deliver as expected, and within the time-boxed cycle, is your own development process, then you can experiment immediately with process improvements, keep them in if they work better. I would assert that this is a superior way of improving development processes compared to the heavy-handed CMMI approaches. We know from client experience that the troops on the ground prefer this approach, because of the empowerment they get.

This evolutionary development process makes it impossible to fail big - just stop, if you are failing in the small increments [Larman, Morris].

The metrics will remind you that you do not know what you are doing! Of course you will probably find that out later, but with *'continuous realistic progress towards critical objectives metrics feedback'* you get the message *sooner*, and might be able to hide your incompetence from your employer or customer for a bit longer.



*Example 3: weekly Timing Prediction improvement at our client Philips Assembléon (2001), using evolutionary methods. The result was delivered directly to customers. The improvement is in ability to correctly predict the real factory run time for chip mounting processes. Previously there had been an over 18-month 'drought' of improvement to customers, while they worked on much larger delivery cycles. "**The results were received with great applause**."*

4. Use metrics to describe metrics, for credibility, and uncertainty.

A metric has its own numeric attributes.

Mertic qualities are for example:

accuracy, credibility, relevance, impact

and costs, like learning cost, test setup cost, test process costs, test analysis costs

We can use metrics to describe and understand our primary metrics.

And to better select both scales of measure, and corresponding measurement processes.

In the Impact Estimation method we can describe each estimate of impact using a set of data:

- the real estimate on the defined scale (like minutes to do a task)
- the ± uncertainty borders of the estimate.
- the % level of impacts with respect to primary targets and deadlines (100% means meets target on time)
- the credibility level of the estimates and uncertainty (scale 0.0 to 1.0)

  which is derived from the parameters
- experience data for making the estimates (facts, case studies, other projects)
- the sources of the experience data (people, groups, papers, studies)

5. Use metrics to describe solutions, designs, and architecture.

All 'designs' have multiple performance/quality/cost attributes, that define 'how well' the designs satisfy our requirements.

In my view it should be normal practice to evaluate all major impact designs, strategies and architectures, against all numeric critical performance and cost targets.

Tools like QFD (Quality Function Deployment) are trying 'structurally', but in my view are not good enough at defining objective real world metrics [Gilb QFD]. Giving a subjective impact evaluation a number like 5, 3 or 7 is not real world metrics – and that is but one small example of my complaint against QFD and similar weak practices. As a consequence, I personally have no confidence whatsoever in QFD evaluations. I agree with one professor, who pleased that, 'surely you would agree it is better than nothing?'

But it is not good enough for the serious problems we all have at hand today. I believe we have more rigorous metrics approaches (like Impact Estimation), even though not as well known as QFD, AHP (Analytic Hierarchy Process), Ralph Keeney's Methods [Keeney] and many other well-meaning attempts to analyze multiple properties of alternative designs.

I wonder how often the obviously-useful engineering tactic, of *numeric multi-dimensional evaluation of a design*, or set of designs, against a set of numeric requirements and constraints, is really used, when it should be, even in non-software forms of engineering. I can personally attest to seeing it missing from most of my work in telecommunications, software, and aircraft product development.

6. Use multiple metrics to compare alternatives.

One way to compare any set of alternatives is to compare their quality and cost attributes,

in relation to your needs (requirements). This is really a special application of the numeric comparison discussed above in principle 5.

This requires the above-discussed capability of defining all your critical performance factors quantitatively – even the apparently soft ones like 'agility', and 'customer orientation', and 'respect in the marketplace'. It then requires the capability of making reasonable, and reasonably based (evidence, sources, credibility as in Impact Estimation) numeric evaluations in relation to those numeric target level requirements. It requires some way of summing up the overall impact on one alternative, on a set of performance and resource requirements (like the % of target, and % of budget concepts in Impact Estimation).

7. Measure critical variables, but with sufficient qualities and lowest costs.

Quantification seems exact: '5.0', '3.14', even though it is perhaps merely an approximation.

Measurement is about determining where we *really* are along a scale of measure, in relation to benchmark levels, constraint levels, and target levels.

Measurement *cannot* be perfect. Perfect measurement has *infinite* cost. Measurement needs to be *sufficient* for purpose, at the lowest costs for that purpose.

Measurement processes can be 'designed' to fit a set of numeric qualities, costs, and constraints.

8. Use metrics to review specifications.

A very basic measure of technical specifications' conformance to their applicable standards, is the number of major defects per volume.

A *defect* is a deviation from a written official applicable standard (like 'R1: All qualities shall have a defined scale of measure.'). A *major* defect is a defect that might potentially cause economic or technical damage at some point (for example something that would have to be reworked before acceptable delivery). The conventional 'volume' I use is 300 non-commentary words of text, a Logical Page. This is not critical, but some *volume standard* needs to be used to normalize the information about defect density.

I usually use a process of sampling random but representative pages of specifications, particularly requirements, and top level requirements, are initially the most interesting.

The process I use, is described in earlier INCOSE lectures [Gilb, SQC, 2005] and [Gilb 2005, CE book, Spec QC chapter].

A small team of experts and managers will usually uncover 20-40 Major defects per page against only 3 standards (clear, unambiguous, no-design in requirements). They are about 33% effective in finding what is actually there at that moment. This can easily be proven by repeating the review process, after the initial batch of defects is removed.

The simple conclusion, that my willing participant managers suddenly grasp, is that they have been approving and accepting highly substandard requirements, and other specs, for years. They are usually immediately ready to discuss how to rewrite the specs to a much cleaner level – like maximum 1.0 majors per page remaining, statistically, and inferred.

9. Use metrics to prioritize, and determine priorities.

I argue that traditional weighting metrics are a very bad way of communicating priorities for requirements.

What are your weights for eating, breathing, drinking?

I would argue that the natural and logical way to understand priorities is in *terms of quantified requirements*, and simultaneously by means of repeated continuous measurement of the *degree of satisfaction of our requirements.*

The more satisfied a requirement, the lower the priority. The more your belly is full, the less hungry you are. [Gilb 2005, and Gilb papers on Priority].

10. Use metrics to create commonly understood, and really agreed requirement or objectives.

It should be obvious that numeric objectives and requirements, increase the probability that people understand and will acknowledge, what they have agreed to.

6.0 is a much clearer notion than ‘very much’

If we agree to ‘extremely good X’

How much have we agreed to?

Summary

We can improve systems engineering by making better use of realistic numbers, instead of subjective opinion numbers (‘5 is medium impact’) or nice sounding words (‘extremely agile’). We should initially use more numbers in describing critical performance requirements, especially soft and quality requirements.

But it is not a matter of simply using numbers, but if using them in a more intelligent way. That means making distinctions amongst different numeric levels of requirements, and clearly specifying the relationships of both requirements and designs to other elements of planning and their environment.

References

Gilb, Tom. ‘QFD, What’s Wrong With Quality Function Deployment’. http://www.gilb.com/community/tiki-download_file.php?fileId=119 Written December 2006, for some systems engineering students at Kongsberg University in Norway, in response to the one-sided propaganda they were getting about the method. Will be submitted to INCOSE 2008.

Gilb, Tom, ‘Competitive Engineering, A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage’, ISBN 0750665076, **2005**, Publisher: Elsevier Butterworth-Heinemann. Sample chapters will be found at Gilb.com.

Chapter 5: Scales of Measure:

http://www.gilb.com/community/tiki-download_file.php?fileId=26

Chapter 10: Evolutionary Project Management:

http://www.gilb.com/community/tiki-download_file.php?fileId=77

Gilb, Tom. SQC, ‘Agile Specification Quality Control: Shifting emphasis from cleanup to sampling defects. INCOSE Proceedings, 2005. Available at www.gilb.com.

Gilb, Tom: (Papers on Priority)
Choice and Priority Using Planguage: http://www.gilb.com/community/tiki-download_file.php?fileId=48 (Not yet accepted by INCOSE)
Managing Priorities (with Mark Maier, INCOSE 2005):
http://www.gilb.com/community/tiki-download_file.php?fileId=60

Gilb.com: www.gilb.com. our website has a large number of free supporting papers , slides, book manuscripts, case studies and other artifacts which would help the reader go into more depth

INCOSE Systems Engineering Handbook v. 3
INCOSE-TP-2003-002-03, June 2006 , www.INCOSE.org,

Ralph Keeney: Value-Focused Thinking - A Path to Creative Decisionmaking. 1996, ISBN 9780674931985.

Larman, Craig, and Vic Basili. "Iterative and Incremental Development: A Brief History". IEEE Computer, June **2003**
http://www2.umassd.edu/SWPI/xp/articles/r6047.pdf

Morris, Peter, The Management of Projects, 1994 Telford, London, 1997 USA.

Biography

BIOGRAPHY

Tom Gilb is an international consultant, teacher and author.

His 9th book is '**Competitive Engineering**: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage' (August 2005 Publication, Elsevier) which is a definition of the planning language 'Planguage'.

He works with major multinationals such as Credit Suisse, Schlumberger, Bosch, Qualcomm, HP, IBM, Nokia, Ericsson, Motorola, US DOD, UK MOD, Symbian, Philips, Intel, Citigroup, United Health, Boeing, Microsoft, and many smaller and lesser known others See www.Gilb.com .

Version 1, 1st draft, completed Sunday 28th Oct 2007, 02:45