2nd International Workshop on Requirements Analysis, London, Middlesex U, Trent Campus
http://www.iwra2010.org/

Invited Keynote by Tom Gilb
Tom@Gilb.com www.gilb.com

Duration (1 hour)
September 2010

Title: **What's Wrong With Requirements Methods?**
Subtitle: *An analysis of the fundamental failings of conventional thinking about software requirements, and some suggestions for getting it right.*

Background:
We know our IT projects fail, and disappoint. We know bad 'requirements' is at the root, but not the only cause. But, it is my opinion that almost no one has openly discussed or dealt with the *real* problem. It is my opinion that no widely-known and widely-taught methods are *anywhere near* identifying the critical problems. In a nutshell: we think like programmers, not engineers or managers. We do not concentrate on value delivery, but instead on code function, on use-cases and on code delivery. Management is not taking its responsibility to make things better.

Outline:
1. Requirement definition: 'Stakeholder Prioritized End State'
2. Ten Reasons Why Requirements Methods Fail
3. Top Level Critical Objectives: the missing link
4. Don't Mix Ends and Means
5. Requirements are not always 'Required': Intelligent Dynamic Prioritization
6. Stakeholders: not just users and customers!
7. Value Delivery: leading to Systems Thinking, not Software Silos
8. Quantification: not 'Software Poetry' – a basis for real Software Engineering – not mere 'Softcrafting'

9. Rich Specification: Requirement specifications need far more info than the requirement itself!
10. Ten Principles for Successful Requirements Methods.
11. Who or What will Change things?

12. Summary

1. Requirement definition: 'Stakeholder Valued End State'

Do we all have a shared notion of what a 'requirement' is?
I am afraid that is one of our problems. Everybody has an opinion, and most of the opinions about the meaning of the concept 'requirement' are at variance with most other opinions. I believe that few of the popular definitions are correct or useful.

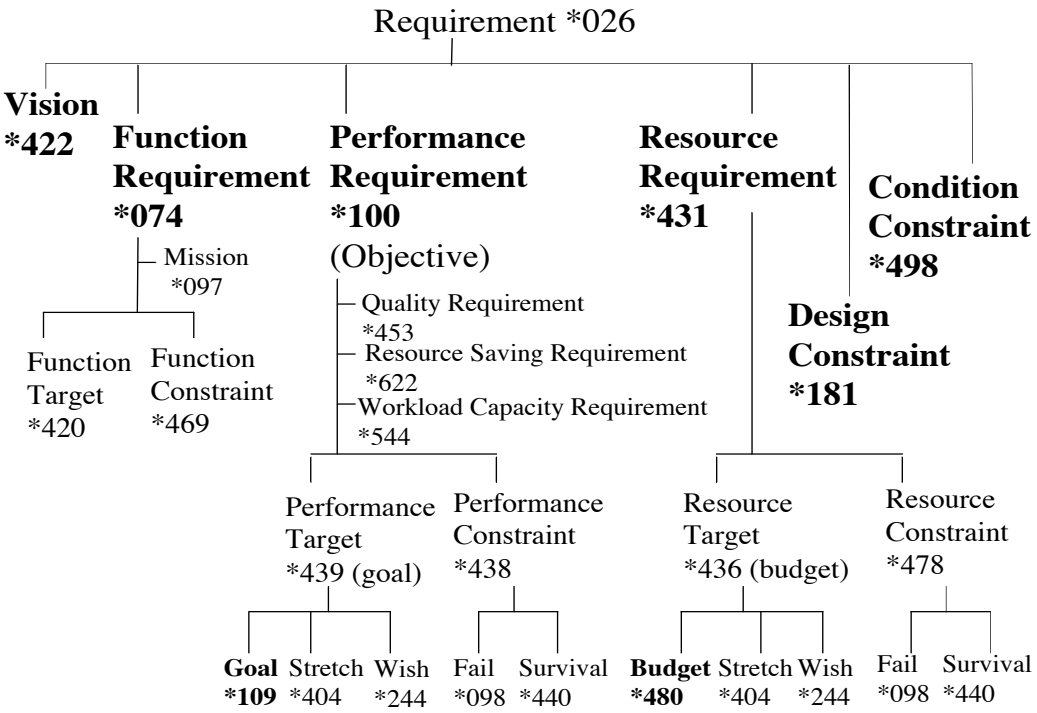
I have spent decades writing, and discussing about requirements. I can reveal my latest 'opinion' about the best definition of 'requirement', but it is possibly not my final definition – perhaps some of you can push me even further.

To get to the point, I have decided to define a 'requirement' as a **"stakeholder-valued end state".**

You probably do not use this, yet, but I will do so in this paper, and I will argue the case.

Notice that I am distinguishing between:

- A 'requirement' (stakeholder-valued end state)
- A requirement specification my Concept *508, [1]
- A an actually implemented requirement
- A 'design' in partial or full service of implementing a requirement.



In addition, I have identified, and defined thoroughly [3, 1] a large number of requirement types and requirement concepts, a sample of which is in the chart above from CE, [3], p.401.

2. Ten Reasons Why Requirements Methods Fail

2.1. They are narrowly focussed on a user or customer need, and not the much broader area of *stakeholder* values.
2.2 They are formulated in terms of designs to satisfy the stakeholder values, but the stakeholder value itself is absent or vague.

- *Example: Password, instead of a Security Level requirement*

2.3 Clearly scalar (variable level) requirements are formulated as nice words, with no attempt to clarify them numerically.

- Example "High Usability"

2.4 Far too much attention to what the system must do (Function) and far too little attention to how well it should do it (qualities) – in spite of the fact that quality improvements tend to be the major drivers for new projects.

2.5 Far too much attention to testable attributes of a system, and too little attention to the **constraint** type of requirement.

- Example: *Constraint:" Must not violate European Union Laws or politics."*

*2.6 Far too much emphasis on the requirement itself; and far too little concurrent information about its **'background'** (*507). Who wants it and why?*

- *Example of Background:*
  - *Availability: Scale: % Planned Uptime.*
    - *Past [Previous Product] 99.90% <- 2010 Study*
    - *Trend [Major Competitors, 2015] 99.99+% <- Mkt Est.*

*2.7 Far too much focus on the individual requirement alone, and far too little on the **valid set** of requirements, needing simultaneous satisfaction*

2.8 Far too little formal and agreed definition of critical or key concepts.

- *BAD: Availability: 99.90%*
- *BETTER: Availability:*
  - *Scale: % Average Planned Uptime Daily for defined [Users] and defined [Tasks]*
  - *Planned: by Operations.*
  - *Uptime: capacity to allow specific Users to Perform specified Tasks at Defined Performance levels.*
  - *Daily: during opening hours*
  - *Goal [Users = Cashiers, Task = Checkout] 99.90%*

*2.9 There is far too little **quality control of requirements**, against decent standards [See 3, 'Rules' for good examples of decent standards] for requirements, before the requirements are exited to next processes.*

- *For example it is typical that we can identify 80 to 200+ words per 300 words of requirement text as ambiguous or unclear to intended readers! [4]*

*2.10 There is far too little systematic work and specification about the related **levels of requirements**. If you look at some methods and processes, all requirements are 'at the same level' (keyword = Burn Rate, Agile Methods).*

- *At the **very least** we need to distinguish between*
  - *Corporate Requirements*
  - *Top Level Critical Few Project or Product Requirements*
  - *Other Stakeholder Requirements*

    - *System Requirements*
    - *Software Requirements*
- *We need to clearly document the level and the relationships between these: for example*
    - ***Availability**:*
        - *Type: Product Line System Level Requirement.*
        - *Stakeholders: {Systems Engineering, Marketing, Architecture}.*
        - *Constrained By: Product Quality Policy, EU Safety Standards*
        - *Impacts: Product Sales, Corporate Image, Competitiveness*
        - *Impacted By: {Software Availability, Data Availability, Network Availability, Server Availability, Handset Availability}.*
    - *By the way, this and earlier examples make use of a **requirements planning language**, I call 'Planguage' [3, 5]. One Planguage convention is that all 'Capitalized Terms' are properly defined, somewhere.*

3. Top Level Critical Objectives [6]: the missing link

I see the 'worst requirement sin of all', in almost *all* projects we look at internationally. The highest level of requirements, the ones that funded the project, are vaguely stated, and ignored by the project team.

They look like this (example below): Each requirement is the top of a page of supporting detail. Most of the detail is a list of suggested technologies that will make this 'wish' come true.

*1. Central to The Corporations business strategy is to be the world's **premier** integrated_<domain> service **provider**.*

*2. Will provide a much more efficient **user** experience*

*3. Dramatically scale back the **time** frequently needed after the last data is acquired to time align, depth correct, splice, merge, recompute and/or do whatever else is needed to **generate** the desired **products***

*4. Make the system much **easier** to **understand** and **use** than has been the case for previous system.*

*5. A primary goal is to provide a much more **productive** system **development** environment than was previously the case.*

*6. Will provide a richer set of functionality for **supporting** next-generation logging **tools** and applications.*

*7. **Robustness** is an essential system requirement (see rewrite in example below)*

*8. Major improvements in **data quality** over current practices*

This is a real example, for a project that used 8 years and over $100 million, without delivering anything of the above requirements.

Note that:

- All requirements are **qualities**, but they were not specified quantitatively, so that they would be crystal clear, and could be tracked.
- Management themselves (CEO, CTO, CIO level) did not take the trouble to clarify these critical objectives. The CEO actively rejected the idea of clarification, the CIO told me!
- None of the technical 'experts' reacted to the situation. They happily spent $100 million on all the many suggested architecture solutions (see Design Hypothesis in the example below for some designs) that were mixed in with the objectives.

It took less than an hour to rewrite one of the objectives so that it was clear, measurable, and quantified. So in one days work they could have clarified the objectives, 8 years of wasted time ago.

Here is the initial stage of the rewrite: Determining that Robustness is complex and composed of many different attributes, such as Testability.

7. Rock Solid **Robustness**:
Type: *Complex* Product Quality Requirement.
Includes: {Software Downtime, Restore Speed, **Testability**, Fault Prevention Capability, Fault Isolation Capability, Fault Analysis Capability, Hardware Debugging Capability}.

And here is one of the Robustness attributes, Testability, defined quantitatively.

Testability:
Type: Software Quality Requirement.
Version: 20 Oct 2006-10-20
Status: Demo draft,
Stakeholder: {Operator, Tester}.
Ambition; Rapid-duration automatic testing of <critical complex tests>, with extreme operator setup and initiation.

Scale: the duration of a defined [Volume] of testing, or a defined [Type], by a defined [Skill Level] of system operator, under defined [Operating Conditions].

Goal [All Customer Use, Volume = 1,000,000 data items, Type = WireXXXX Vs DXX, Skill = First Time Novice, Operating Conditions = Field, {Sea Or Desert}. <10 mins.

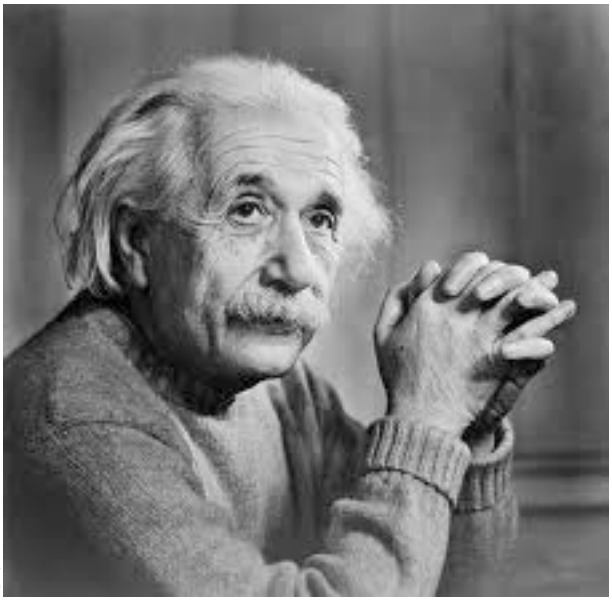
Design Hypothesis: *Tool Simulators, Reverse Cracking Tool, Generation of simulated telemetry frames entirely in software, Application specific sophistication, for drilling – recorded mode simulation by playing back the dump file, Application test harness console <-6.2.1 HFA*

Notice:

- Management has lost control at the beginning
- But it takes 8 years delay and a loss of $100 million to make management aware something is wrong (must be those programmers, right?)
- The technical staff is no better, and happily spends management's money. *"A fool and their money will soon part company, for they have more dollars than sense (variant, Tussler)"*
- *Neither managers nor nerds have culture or training to quantify qualitative requirements. They act like nice-sounding words are good enough.*

- *There was, I found, a 3 cm. high stack of paper with some very detailed 'requirements', and 80 developers, but nobody had ever delivered what management wanted 8 years earlier.*

4. Don't Mix Ends and Means

"Perfection of means and confusion of ends seem to characterize our age."	Albert Einstein. 1879-  1955

The problem of confusing ends and means is clearly an old one, and deeply rooted.
We specify a solution, design, architecture, instead of what we really value – our real requirement [7].

There are explanatory reasons for this – for example solutions are more concrete, and what we went (qualities) are more abstract for us (because we have not learned to make them measurable and concrete).

The problem is, if we do confuse them: if we do specify the means, not our true ends

- "be careful what you ask for, you might just get it" (unknown source)
- you might not get what you *really* want
- the solution you have specified might *cost too much* or have bad *side effects*, even if you do get what you want
- there may be much *better solutions* you don't know about yet

So how to we find the 'right requirement', the 'real requirement'? [7]

- *Assume* that there probably is a better formulation, a more accurate expression of our real values and needs
- *Search for it by asking 'Why?'*
    - Why do I want X, because I really want Y, and assume I will get it through X. But, then why do I want Y? Because I really want Z and assume that is the best way to get X. Continue the process until it seems reasonable to stop. (the 5 Whys' Method)
- Assume that our stakeholders will *usually* state their values in terms of some perceived means to get what they really value.
    - Help them identify (Why? Process) and acknowledge what they really want, and make that the 'official' requirement.
    - Don't insult them by telling them they don't know what they want.
    - But explain that you will help them more-certainly get what they more deeply want, with better and cheaper solutions, perhaps new technology, if they will go through the 'Why?' process with you

- 'Why do you require a 'password'?
  - for Security!
- What kind of security do you want?
  - Against stolen information
- How strong security against stolen info are you willing to pay for.
  - At least 99% chance they cannot break in within 1 hour of trying! Whatever that costs up to € 1 million.
- So that is your real requirement ?
  - Yep.
- Can we make that the official requirement, and leave the security design to both our security experts, and leave it to proof by measurement to decide what is really the right design?
  - Of course!

Description of requirement/work task	Past	Status
Usability.Productivity: Time for the system to generate a survey	7200 sec	15 sec
Usability.Productivity: Time to set up a typical specified Market Research-report (MR)	65 min	20 min
Usability.Productivity: Time to grant a set of End-users access to a Report set and distribute report login info.	80 min	5 min
Usability.Intuitiveness: The time in minutes it takes a medium experienced programmer to define a complete and correct data transfer definition with Confirmit Web Services without any user documentation or any other aid	15 min	5 min
Performance.Runtime.Concurrency: Maximum number of simultaneous respondents executing a survey with a click rate of 20 sec and an response time<500 ms, given a defined [Survey-Complexity] and a defined [Server Configuration, Typical]	250 users	6000

Case study [9] The best improved 5 of 25 quantified required results in 3 months of development. A client of ours, who directly specified as requirements, what they really wanted (like 'intuitiveness') and then quantified it, making it fairly simple to measure after each iterative week of Evo method development. They did not make the mistake of letting the 'users' design the system. In fact they consciously REFUSED to allow such 'requirements' from their customers! They let the developers do the design, and let developers measure the results of their designs. They got so good so fast that they wiped out, bought up, their competition.

7. Value Delivery: leading to Systems Thinking, not Software Silos

Requirements are 'stakeholder valued future states'. The whole point is VALUE. Not function, not stories. The whole point is value to STAKEHOLDERs, not just *users* or *customers* alone!

Value (*269, [1]) is:
"Value is *perceived* benefit: that is, the benefit we think we get from something." [3, page 435]

One problem with conventional requirements thinking is that it is not closely enough coupled with 'value'.

If requirements are NOT closely tied to value then:

- We risk failure to deliver the value expected, even if 'requirements' are satisfied.
- We risk having a failure to think about all the things to do that are necessary prerequisites to actually delivering *full value* to real *stakeholders* on time, *systems* thinking – not just programming.

We need to keep in mind that we *'softcrafters'* [2] *('engineers', is too pretentious a title, for such unpredictable and poor results)* have a terrible failure rate for IT projects (Standish Chaos report etc.). Part of the reason for this, many analysts agree, is the poor state of requirements methods and practice. *Failure* is 'failure to deliver expected value'.

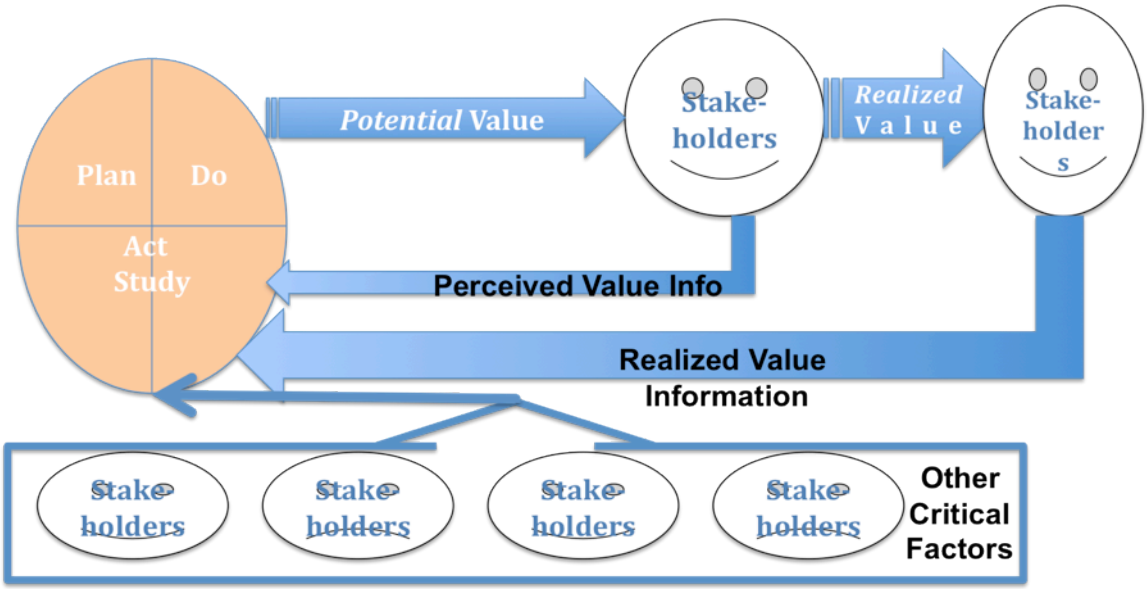


Figure: Value can be delivered gradually, incrementally, to stakeholders. (Diagram © Tom Gilb, 2009).

Stakeholders can give feedback about their perception of value, based on *realities*. The whole process is a Plan Do Study Act Learning process involving many and complex factors, including factors from outside the system, such as politics, law, international differences, economics, and technology change. The 'requirements' must be *evolved* based on realistic experience. Attempts to fix them in advance, of this experience flow, are probably wasted energy, if they are committed to – in contracts and fixed specifications. Notice this diagram makes the subtle distinction between initially perceived value ('I think that would be useful'), and effective and factual value ('this was in practice more valuable than we thought it would be, because …').

How can we articulate and document notions of value in a requirement specification?
Here is a sketch using Planguage [3].

Usability.Intuitiveness:
Type: Marketing Product Requirement.
Stakeholders: Marketing Director, Support Manager, Training Center
Impacts: Product Sales, Support Costs, Training Effort, Documentation Design.
Supports: Corporate Quality Policy 2.3

Ambition: Any potential user, any age, can immediately discover and correctly use all functions of the product, without training, help from friends, or external documentation
Scale: % chance that defined [User] can successfully complete defined [Tasks] Immediately, with no External help.
Meter [Consumer Reports] tests all tasks for all defined user types, and gives public report.
---------- Analysis ----------
Trend [Market = Asia, User = {Teenager, Early Adopters}, Product = Main Competitor, Projection = 2013] 95%±3% <- Market Analysis
Past [Market = USA, User = Seniors, Product = Old Version, Task = Photo Tasks Set, When = 2010] 70% ±10% <- Our Labs Measures
Record [Market = Finland, User = {Android Mobile Phone, Teenagers}, Task = Phone+SMS Task Set, Record Set = January 2010] 98% ±1% <- Secret Report
---------- Our Product Plans ----------
Goal [Market = USA, User = Seniors, Product = New Version, Task = Photo Tasks Set, When = 2012] 80% ±10% <- Draft Marketing Plan
Tolerable [Market = Asia, User = {Teenager, Early Adopters}, Product = Our New Version, Deadline = 2013] 97%±3% <- Mkt Dir. Speech
Fail [Market = Finland, User = {Android Mobile Phone, Teenagers}, Task = Phone+SMS Task Set, Product Release 9.0] Less Than 95%

Practical made-up example, designed to display ways of making the value of a requirement clear.

It would be too much in this paper to explain every detail, but hopefully the patient reader can deduce a great deal. The Competitive Engineering text [3] is the detailed definition of these concepts. I would assert that all details here arguably document some notion of the value of the requirement.

I will give some examples, so you get the spirit of things.

Decoding a single statement – where is the value delivered?

The Goal (80%) specifies which market (USA) it is intended for, which set of tasks are valued (the 'Photo Tasks set'), and when it would be valuable to get it delivered (2012). This 'qualifier' information in all the statements, helps document when, where, who, what, when the value of the quality level applies to.

Relating One Statement to Others, regarding value.

The Record statement gives information about the known state of the art. So any requirement to beat that (Goal, Tolerable, Fail) would be saying there is value in being the best. In this case the plan is to avoid being significantly worse (Fail level is set at less than 95%). There does not seem to be any value worth paying for in beating the record.

Hopefully no you can figure out the other value, and relative value, assertions. A more common specification of requirement, that we often see, like:

"2.4 The product will be more user-friendly, using Windows"

is unclear, pre-empts good design, and has no information about value whatsoever.

So who is going to make these value statements in requirements specifications?

I don't expect developers to care much about value statements in requirements. Their job is to deliver the requirement levels that someone has determined are valued. Deciding what is valuable to require is a Product Owner (Scrum) or Marketing Management function. We developed a training course for them [10].

8. Quantification: not 'Software Poetry' – a basis for real Software Engineering – not mere 'Softcrafting'

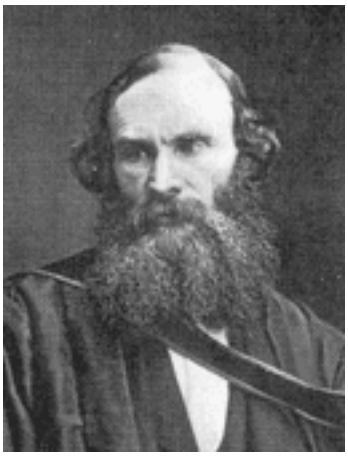
Some developers call themselves 'software engineers', they might even have a degree in the subject, or in 'computer science'. But they do not seem to practice any real engineering [11]. When I look at what they do and what they are taught, I think it looks more like 'software poetry'.

Nice sounding words; good enough to fool managers into spending millions for nothing! [6,7] Wow, that is effective seductive poetry!

Engineering is a practical bag of tricks. My dad was a real engineer (100 patents too!) and I don't remember him using poetry. He seemed forever there with slide rules and back-of-the-envelope calculations. Whatever he did, he could you tell why it was numerically superior to somebody else's

product. He argued with numbers and measures. Not impressive sounding words. AKA Management BS
[http://www.julianwellings.com/buzzwords4u/]

My life changed professionally, when, in my Twenties, I read the words of the Lord,

(Lord Kelvin) 

> "*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.*" *[PLA, vol. 1, "Electrical Units of Measurement", 1883-05-03]*

or, also credited Lord Kelvin:

> "*If you can not measure it, you can not improve it.*"

The most frequent and critical reasons for software projects are to improve them qualitatively compared to their predecessors (which may or may not be automated logic).

But, we seem to have totally avoided the practice of quantifying these qualities, so as to make them clearly understood, and also so as to lay the basis for measuring our progress in improvement towards our quality level requirements.

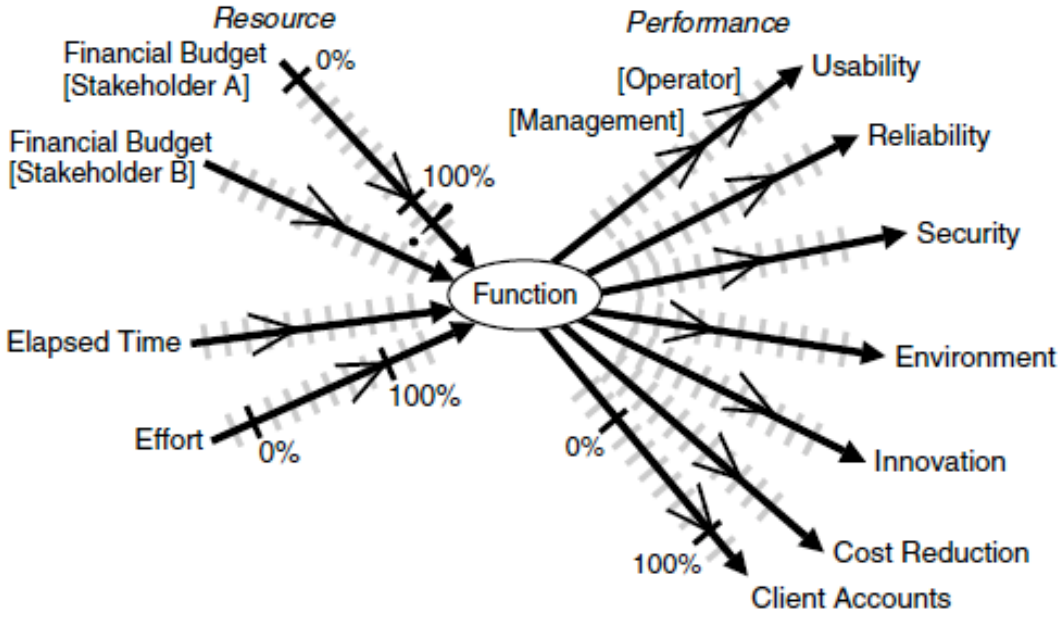
This art of quantification of any quality requirement should be taught as fundamental to university students of management and software disciplines. It seems to be somewhat more taken for granted in other sciences and engineering. One problem seems to be that the teachers themselves do not appreciate quality dimensions and requirements are numeric, and cannot teach it.

The problem is not that managers and software people cannot and do not quantify. They do. It is the lack of 'quantification of the qualitative' that is the problem.

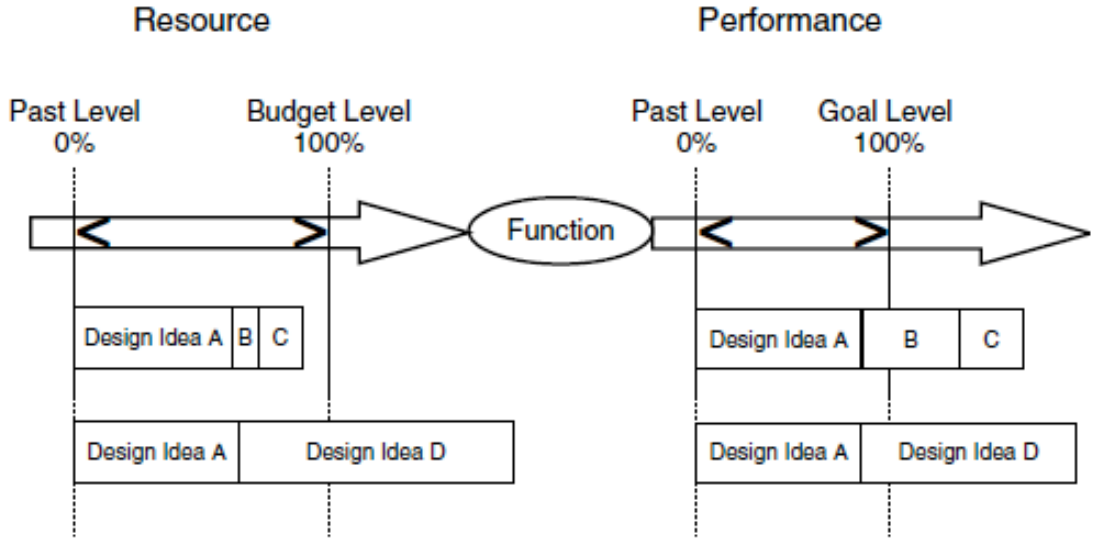
Perhaps we need an agreed definition of 'quality'/'qualitative' before we proceed, since the common interpretation is too narrow, and not well agreed. Most software developers when they say 'quality' are only thinking of bugs (logical defects) and little else. Managers speaking of the same software do not have a broader perspective. They speak and write often of qualities, but do not usually refer to the broader set of '-ilities' as qualities, unless pressed to do so. They may speak of improvements, benefits instead.

I believe that the concept of 'quality' is simplest explained as 'how well something functions'. I prefer to specify that it is necessarily, since there are degrees of how well, a 'scalar' attribute [3, 1].

This contrasts with other requirement-related concepts such as Capacity (how much performance), Cost (how much resource), Function (what we do), design (how we might do function well, at a given cost) [3,1]



*Source [3] p. 163. A way of visualizing qualities in relation to function and cost. Qualities, like costs, are scalar variables, so we can define scales of measure in order to discuss them numerically. The →-- arrows on the scales represent interesting points such as **requirement** levels. The requirement is not, security, but a defined and provable degree of security.*



Source [3] p. 192. A graphical way of understanding performance attributes (which include all qualities) in relation to function, design and resources. Design ideas cost some resources, and design ideas deliver performance for given functions.

My detailed argument about quantification is freely downloadable Chapter, Scales of Measure, Chapter 5 [3b].

My simple belief is that absolutely all qualities that we value in software (and associated systems) can be expressed quantitatively. I have yet to see an exception. Of course most of you do not know that, or believe it.

The simplest way to explore this for yourself is to search the internet.

For example:
"*Intuitiveness* scale measure": turns up 3 million hits,
Including this excellent study [12], (adding 'Gilb' reduces results to about 952 hits ☺)

Several major corporations have top level policy, sometimes suggested by me, sometimes just because they are good engineers, to quantify all quality requirements. They include IBM, HP, Ericsson, and in practice Intel (who uses Planguage extensively, [3]).

The key idea for quantification is to define, or reuse a definition, of a scale of measure:

For example: (earlier given with more detail)

Usability.**Intuitiveness**:
Type: Marketing Product Quality Requirement.
Ambition: Any potential user, any age, can immediately discover and correctly use all functions of the product, without training, help from friends, or external documentation

Scale: % chance that defined [User] can successfully complete defined [Tasks] Immediately, with no External help.

Meter [Consumer Reports] tests all tasks for all defined user types, and gives public report.

Goal [Market = USA, User = Seniors, Product = New Version, Task = Photo Tasks Set, When = 2012] 80% ±10% <- Draft Marketing Plan

A simple example of quantifying a quality requirement, 'Intuitiveness'.

Explanation of the example above.
1. **Usability.Intuitiveness** is a cross reference name tag. A hierarchy of defined Usability requirements is implied. One subset of Usability is Intuitiveness. The Capital letters indicate formally defined concepts.
2. **Type**: the declaration 'Quality Requirement' by our definitions, demand that we define this numerically (using Scale, or other devices).
3. **Ambition**: is a high level 'poetic' summary of the requirement. One that is easy to agree to, and understand roughly. The Scale and Goal following it MUST correlate to this Ambition statement.
4. **Scale**: is the formal definition of our chosen scale of measure. The parameters [User] and [Task] allow us to generalize here, while becoming more-specific in detail below (see earlier example). They also encourage and permit reuse of the Scale, a sort of 'Pattern'.
5. **Meter**: is a defined measuring process. There can be more than one for different occasions. Notice the Kelvin quotation above, how he twice in the same sentence distinguishes carefully between numeric definition (scale), and measurement process or instrument (Meter). Many people, I hope you are not one, think they are the same thing. Km/hour is NOT a Speedometer. A Voltmeter is not a Volt.
6. **Goal:** is one of many possible requirement levels (see earlier detail for some others; Fail, Tolerable, Stretch, Wish, are other requirement levels). We are defining a **stakeholder valued future state** (state = 80% ± 10%).

One ***stakeholder*** is 'USA Seniors'. The ***future*** is 2012. The requirement level type 'Goal' is defined [1,3] as a very high priority, promise of delivery. Higher than a Stretch or Wish level. But, other priorities may conflict and prevent this particular requirement from being delivered in practice.

If you know the *conventional* state of requirements methods, then you will now, from this example alone, begin to appreciate the difference that I am proposing. Especially for *quality* requirements. I know you can quantify time, costs, speed, response time, burn rate, and bug density – but there is *more*!

But most of the arguments and ideas I am putting forth apply to all types of requirements, the exception being that scalar quantification does not

apply to *binary* requirements like function (to Draw, to Erase) and *binary* conditions (be Legal, be Invisible) [3] for detail.

Any practice of requirements, that does not
1. acknowledge quality requirements as a critical and major type of requirement
2. quantify all quality requirements
is very immature and dangerous, for non-trivial projects.

Teachers, writers, consultants, and learning institutions who do not deal with this 'quality quantification' are, in my opinion, also dangerous to our community. They become part of why our projects fail.

9. Rich Specification [13]: Requirement specifications need far more info than the requirement itself!

There is a persistent bad habit in requirements methods and practices. We seem to specify the 'requirement itself', and we are finished with that specification. I think our requirement specification job might be less than 10% done with the 'requirement itself'.

I believe that it is absolutely necessary, and should be a corporate standard, to include a great deal of other related information, intimately and immediately tied into the spec of the requirement itself. I have a word for this, 'Background' specification.

Background **Concept *507 May 26, 2003 C**
Background information is the part of a specification, which is *useful* related information, but is not *central* (*core)* to the implementation, nor is it commentary.

Examples:

Background parameters include:
- Benchmarks {Past, Record, Trend}
- Owner
- Version
- Stakeholders
- Gist

- Ambition
- Impacts
- Supports

And, also

Commentary **Concept *632 May 26, 2003**

Commentary specifications are remarks about other specifications. Commentary specifications will probably not have any economic, quality or effort consequences if they are incorrect: defects in commentary are almost always of minor severity.

Examples:

- Note
- Comment
- "Text in quotes"
- Source

by contrast;

Core Specification **Concept *633 May 26, 2003 D**

Anything classed as, 'core specification,' will result in real system changes being made: incorrect core specification would materially and negatively affect the system in terms of costs, effort or quality. Specification defects in core specification are almost always of major severity.

Examples:

Core Specification Parameters include:

- Scale
- Meter
- Goal
- Definition
- Constraint

So, why do the background specification elements need to be included?

Here are some function of the background information:

1. help judge value of the requirement
2. help prioritize the requirement
3. help understand risks with the requirement
4. help present the requirement in mire or less detail to various audiences and purposes
5. to give us help when updating a requirement
6. to synchronize the relationships between different but related levels of the requirements
7. to assist in quality control of the requirements
8. to improve the clarity of the requirement
9. any many more purposes..

Here is an example which illustrates some tactics for point 3, **risks**:

Reliability:
Type: Performance.Quality.
Owner: Quality Director
Author: John Engineer
Stakeholders: {Users, Shops, Repair Centers}.
Scale: Mean Time Between Failure.
Goal [Users]: 20,000 hours. <- Customer Survey, 2004
Rationale: anything less would be uncompetitive.
Assumption: our main competitor does not improve more than 10%.
Issues: new competitors might appear.
Risks: the technology for reaching this level might have excessive costs.
Design Suggestion: triple redundant software and database system.
Goal [Shops]: 30,000 hours. <- Dixons' Chain [Quality Director].
Rationale: customer contract specification.
Assumption: this is technically possible today.
Issues: the necessary technology might cause undesired schedule delays.
Risks: the customer might merge with a competitor chain and leave us to foot the costs that they might no longer require.
Design Suggestion: Simplification and reusing known components.

Example: a requirement specification can be embellished with many background specifications that will help us to understand risks associated with one or more other requirement specification elements.

Source: [13]

Let me emphasize that I do not believe that this background information is sufficient if it is scattered around in different documents and meeting

notes. I believe it needs to be directly integrated into a master sole reusable requirement specification object for each requirement. Otherwise it will not be available when it is needed, and will not be updated, or shown to be inconsistent with emerging improvements in the requirement specification.

Here is a requirement template which will hint at the richness possible: Source: [3] Competitive Engineering, Function chapter.

Template for Function Specification <with hints>

Tag: <Tag name for the function>.
Type: <{Function Specification,
Function (Target) Requirement,[6]
Function Constraint}>.
========== **Basic Information** ==========
Version: <Date or other version number>.
Status: <{Draft, SQC Exited, Approved, Rejected}>.
Quality Level: <Maximum remaining major defects/page, sample size, date>.
Owner: <Name the role/email/person responsible for changes and updates to this specification>.
Stakeholders: <Name any stakeholders with an interest in this specification>.
Gist: <Give a 5 to 20 word summary of the nature of this function>.
Description: <Give a detailed, unambiguous description of the function, or a tag reference to some place where it is detailed. Remember to include definitions of any local terms>.
========== **Relationships** ==========
Supra-functions: <List tag of function/mission, which this function is a part of. A hierarchy of tags, such as A.B.C, is even more illuminating. Note: an alternative way of expressing supra-function is to use Is Part Of>.
Sub-functions: <List the tags of any immediate sub-functions (that is, the next level down), of this function. Note: alternative ways of expressing sub-functions are Includes and Consists Of>.
Is Impacted By: <List the tags of any design ideas or Evo steps delivering, or capable of delivering, this function. The actual function is NOT modified by the design idea, but its presence in the system is, or can be, altered in some way. This is an Impact Estimation table relationship>.
Linked To: <List names or tags of any other system specifications, which this one is related to intimately, in addition to the above specified hierarchical function relations and IE-related links. Note: an alternative way is to express such a relationship is to use Supports or Is Supported By, as appropriate>.
========== **Measurement** ==========
Test: <Refer to tags of any test plan or/and test cases, which deal with this function>.
========== **Priority and Risk Management** ==========
Rationale: < Justify the existence of this function. Why is this function necessary? >.
Value: <Name [Stakeholder, time, place, event>]: <Quantify, or express in words, the value claimed as a result of delivering the requirement>.
Assumptions: <Specify, or refer to tags of any assumptions in connection with this function, which could cause problems if they were not true, or later became invalid>.
Dependencies: <Using text or tags, name anything, which is dependent on this function in any significant way, or which this function itself, is dependent on in any significant way>.
Risks: <List or refer to tags of anything, which could cause malfunction, delay, or negative impacts on plans, requirements and expected results>.
Priority: <Name, using tags, any system elements, which this function can clearly be done *after* or must clearly be done *before*. Give any relevant reasons>.
Issues: <State any known issues>.
========== **Specific Budgets** ==========
Financial Budget: <Refer to the allocated money for planning and implementation (which includes test) of this function>.

10. Ten Principles for Successful Requirements Methods.

Here is a summary of my advice for more successful requirement methods in the form of some principles, or 'admonishments':

1. Quality requirements must be quantified.
2. Requirement specifications must be rich with relevant background
3. Requirements must be finally developed based on incremental feedback from stakeholders, as to their real value
4. Requirements need to be accompanied by many types of signals about their priority, and value
5. Requirements must represent the stakeholders' real and core values, not a perceived means of delivering those values
6. The top-level most-critical-few project requirements, are the major focus; all others are supporting details
7. Requirements are not 'required': they are merely *valued*
8. The top ten critical requirements for any project can be quantified and put on a single page.
9. A good first draft of the top ten critical requirements for any project can be made in a day's work.
10. Requirements will forever change, because our world is changing, so don't ask to get final stable requirements from anyone ever.

11. Who or What will Change things?

Everybody talks about requirements, but nobody does anything about it.

I am a pessimist. The majority of IT shops we encounter are not highly motivated to learn or practice first class requirements (or anything else!). Their managers are not motivated, the developers are not.

They get by with, and get well paid for, failed projects.

There are serious internationally competitive businesses like HP and Intel that have long since improved their practices because of their competitive nature and necessity. But they are very different from the majority of those building software.

People will do what they are told to do, trained to do, and managed to do by their employers: but their employers seem largely unconscious.

The knowledge is there, freely available on the web for example, but people cannot bother getting trained and applying the discipline. Why should they? They get by.

The fashion now is to learn oversimplified methods, and methods prescribed by some certification or standardization body. Interest in learning provably more-effective methods is left to the enlightened and ambitions few – as usual.

The technical universities certainly do not train people well in requirements, and the Business Schools certainly do not train managers about this [14].

So, the only interesting game in town are the elite few organizations and individuals who do in fact realize the competitive edge they get with better practices [6, 9].

Maybe this is simply the way the world is. First Class and real Masters of the art are rare. Sloppy 'Muddling through' is the norm. Failure is inevitable or denied.

Insurance companies and lawmakers might demand better practices, but I fear that even *that* would be corrupted in practice, if history is any guide (think CMMI and US DoD, and Indian IT all at level 5 ☺).

I am sitting here writing with the BP Gulf Oil Leak Disaster in mind – excuse my pessimism! The BP CEO Hayward just got his reward today of £11 million in pension rights for managing the Oil Spill and 11 deaths. He once said his main job ('requirement' ?) was "to focus 'laser like' on

safety and reliability" (2007). I wonder what that MBS means to him? Seems the worse you do, the more you get paid?

Welcome if you want to be exceptional! I'd be happy to help!

12. Summary

Requirement methods are woefully inadequate for today's critical and complex systems. There seems to be wide agreement about that.

We know what to do, if we want to, and some corporations have done so, some projects have done so, some developers have done so, some professors have done so: but when is the other 99.99 % of requirements stakeholders going to wake up and do things to a decent standard?

I have personally seen several real projects where the executives involved allowed over $100 million to be wasted on software projects, rather than ever changing their corporate practices.

$100 million here and there, corporate money, is not big money to these guys! Reminds me of politicians sending millions of soldiers to fight for glory. Reminds me of Wall Street 2007.

However, if there be some executives or Boards, governments, or professors or consultancies, who want to try to improve things with project requirements – we know what to do about it.

References:

[1] Planguage Concept Glossary:

- All glossary concepts are identified by a number '*nnn'
- [1a] An edited subset (10%) is in [3] the CE book
  - http://www.gilb.com/tiki-download_file.php?fileId=387
- [1b] The unabridged and periodically updated Planguage glossary
  - http://www.gilb.com/tiki-download_file.php?fileId=386
  - At the moment the March 2010 edition is there, and the Requirement concept is not updated.
- [1c] A Paper: A Conceptual Glossary for Systems Engineering: *Define the Concept, don't quibble about the terms. 2006. INCOSE Paper.*
  - http://www.gilb.com/tiki-download_file.php?fileId=125
-

[2] Gilb, Tom, Principles of Software Engineering management (chapter 3, 8, 9 on Requirements), Addison-Wesley, 1988. http://www.amazon.com/Principles-Software-Engineering-Management-Gilb/dp/0201192462

[3] [3a]Gilb, Tom. Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering, Using Planguage, 2005, Elsevier. ISBN 0 7506 6507 6. http://www.amazon.com/dp/0750665076/ref=rdr_ext_sb_ti_sims_2
[3b] Chapter 5, Scales of Measure, http://www.gilb.com/tiki-download_file.php?fileId=26
[3c] Chapter 10 Evo, http://www.gilb.com/tiki-download_file.php?fileId=77

[4] Specification Quality Control

- [4a] see [3] Chapter 8, Specification Quality Control
- [4b] Gilb: "Agile Specification Quality Control", in Testing Experience, March 2009
  - http://www.gilb.com/tiki-download_file.php?fileId=264
  - www.**testingexperience**.com/**testingexperience**01_08.pdf
    - **Note this link needs testing!**

[5] AGILE ASPECTS OF PLANGUAGE: For Cost-Effective Engineering. 2006 Gilb Paper.

- **http://www.gilb.com/tiki-download_file.php?fileId=39&highlight=agile%20planguage**

[6] Top Level Objectives: A slide collection of case studies

- http://www.gilb.com/tiki-download_file.php?fileId=180

[7] Gilb: Real Requirements

- http://www.gilb.com/tiki-download_file.php?fileId=28

[8]]. Eliyahu M. Goldratt, The Choice, North River Press, 2008. This discusses root cause analysis, among other subjects. http://www.northriverpress.com/product50.html

[9] Confirmit case: a Norwegian survey software and service company. Using our Evo method excellently since 2003. http://www.gilb.com/tiki-download_file.php?fileId=32

[10] Value Planning Slides for Scrum Product Owners http://www.gilb.com/tiki-download_file.php?fileId=353

[11] Dr. Billy Vaughn Koen, DISCUSSION OF THE METHOD Conducting the Engineer's Approach to Problem Solving Oxford University Press, March, 2003. http://www.cse.hcmut.edu.vn/~minhle/congtackysu_2008/Engineering_Method.pdf The book is a great in depth tour de force, the URL is to a summary paper. This is my favorite discussion about what is 'engineering'?

[12] Zhilin Yanga,*, Shaohan Caib, Zheng Zhouc, Nan Zhoua "Development and validation of an instrument to measure user perceived service quality of information presenting Web portals", Information & Management 42 (2005) 575–589 http://www.fbe.hku.hk/~kevinzhou/5%20Web%20Portal.pdf

[13] Gilb, "Rich Requirement Specs: The use of Planguage to clarify requirements". http://www.gilb.com/tiki-download_file.php?fileId=44

[14] Hopper: The Puritan Gift, http://www.puritangift.com/ Wonderfully deep history of management practices and why they got corrupted with the Business Schools. Useful Blog