J. Software Engineering & Applications, 2010, 3, 827-838
doi:10.4236/jsea.2010.39096 Published Online September 2010 (http://www.SciRP.org/journal/jsea)

What's Wrong with Requirements Specification? An Analysis of the Fundamental Failings of Conventional Thinking about Software Requirements, and Some Suggestions for Getting it Right

Tom Gilb

Result Planning Limited, Norway and UK.
Email: Tom@Gilb.com

ABSTRACT

We know many of our IT projects fail and disappoint. The poor state of requirements methods and practice is frequently stated as a factor for IT project failure. In this paper, I discuss what I believe is the fundamental cause: we think like programmers, not engineers and managers. We do not concentrate on value delivery, but instead on functions, on use-cases and on code delivery. Further, management is not taking its responsibility to make things better. In this paper, ten practical key principles are proposed, which aim to improve the quality of requirements specification.

Keywords*: *Requirements*, *Value Delivery*, *Requirements Definition*, *Requirements Methods

1. Introduction

We know many of our IT projects fail and disappoint. We know bad 'requirements', that is requirements that are ambiguous or are not really needed, are often a factor. However in my opinion, the real problem is one that almost no one has openly discussed or dealt with. Certainly, it fails to be addressed by many widely known and widely taught methods. So what is this problem? In a nutshell: it is that we think like programmers, and not as engineers and managers. In other words, we do not concentrate on value delivery, but instead on functions, on use cases and on code delivery. And no one is attempting to prevent this: IT project management and senior management are not taking their responsibility to make things better.

2. Ten Key Principles for Successful Requirements

In this paper, my ten key principles for improving the approach to requirements are outlined. These principles are not new, and they could be said to be simply commonsense. However, many IT projects still continue to fail to grasp their significance, and so it is worth restating them. These key principles are summarized in **Figure 1**. Let's now examine these principles in more detail and provide some examples.

Note, unless otherwise specified, further details on all aspects of Planguage can be found in [1].

2.1. Understand the Top Level Critical Objectives

I see the 'worst requirement sin of all' in almost *all* projects we look at, and this applies internationally. Time and again, the high-level requirements (the ones that funded the project), are vaguely stated, and ignored by the project team. Such requirements frequently look like the example given in **Figure 2**.

The requirements in **Figure 2** have been slightly edited to retain anonymity. They are for a real project that ran for eight years and cost over 100 million US dollars. The project failed to deliver any of these requirements. However, the main problem is that these are not top-level requirements: they fail to explain in sufficient detail what the business is trying to achieve. There are additional problems as well that I'll discuss further later in this paper (such as lack of quantification, mixing optional designs into the requirements, and insufficient background

Ten Key Principles for Successful Requirements

1 Understand the top level critical objectives
2 Look towards value delivery: systems thinking, not just software
3 Define a 'requirement' as a 'stakeholder-valued end state'
4 Think stakeholders: not just users and customers!
5 Quantify requirements as a basis for software engineering
6 Don't mix ends and means
7 Focus on the required system quality, not just its functionality
8 Ensure there is 'rich specification': requirement specifications need far more information than the requirement itself!
9 Carry out specification quality control (SQC)
10 Recognize that requirements change: use feedback and update requirements as necessary

Figure 1. Ten key principles for successful requirements.

Example of Initial Top Level Objectives

1 Central to the corporation's 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, recomputed 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 with the previous system
5 A primary goal is to provide a much more productive system development environment then 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
8 Major improvements in data quality over current practices

Figure 2. Example of initial top level objectives.

description).

Management at the CEO, CTO and CIO level did not take the trouble to clarify these critical objectives. In fact, the CIO told me that the CEO actively rejected the idea of clarification! So management lost control of the project at the very beginning.

Further, none of the technical 'experts' reacted to the situation. They happily spent $100 million on all the many suggested architecture solutions that were mixed in with the objectives.

It actually took less than an hour to rewrite one of these objectives so that it was clear, measurable, and quantified. So in one day's work the project could have clarified the objectives, and avoided 8 years of wasted time and effort.

1) The top ten critical requirements for any project can be put on a single page.

2) A good first draft of the top ten critical requirements for any project can be made in a day's work, assuming access to key management.

2.2. Look towards Value Delivery: Systems Thinking, not Just a Focus on Software

The whole point of a project is delivering realized value, also known as benefits, to the stakeholders: it is not the defined functionality, and not the user stories that count. Value can be defined as the benefit we think we get from something [1]. See **Figure 3**. Notice the subtle distinction between initially perceived value ('I think that would be useful'), and realized value: effective and factual value ('this was in practice more valuable than we thought it would be, because …').

The issue is that conventional requirements thinking is that it is not closely enough coupled with 'value'. IT business analysts frequently fail to gather the information supporting a more precise understanding and/or the calculation of value. Moreover, the business people when stating their requirements frequently fail to justify them

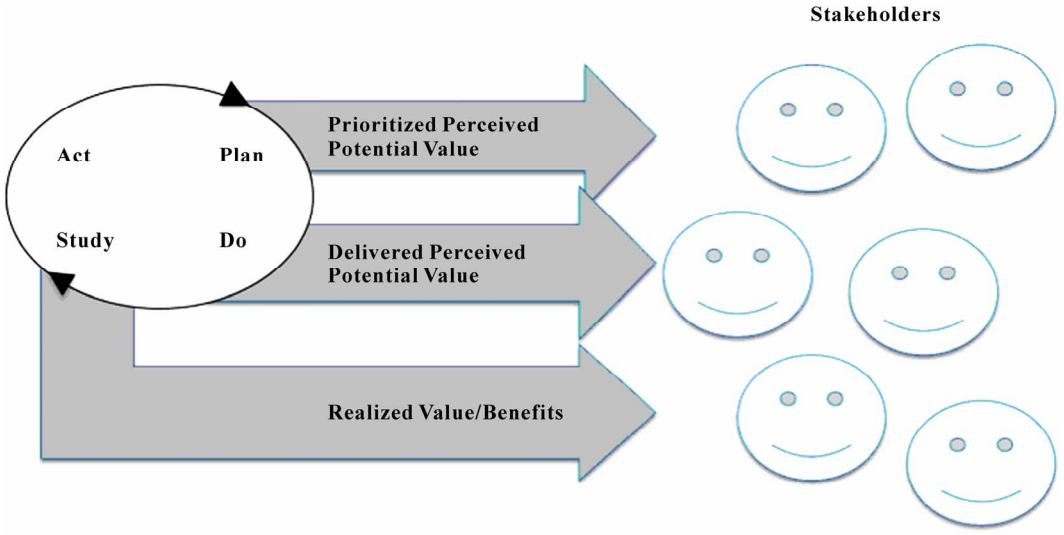

Figure 3. Value can be delivered gradually to stakeholders. Different stakeholders will perceive different value.

using value.

The danger if requirements are not closely tied to value is that:

1) We risk failure to deliver the value expected, even if 'requirements' are satisfied

2) 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: we need *systems* thinking – not just programming.

How can we articulate and document notions of value in a requirement specification? See the Planguage example for Intuitiveness, a component quality of Usability, in **Figure 4**.

For brevity, a detailed explanation is unable to be given here. Hopefully, the Planguage specification is reasonably understandable without detailed explanation. For example, the Goal statement (80%) specifies which market (USA) and users (Seniors) 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 where, who, what, and when the quality level applies. The additional Value parameter specifies the perceived value of achieving 100% of the requirement. Of course, more could be said about value and its specification, this is merely a 'wake-up call' that explicit value needs to be captured within requirements. It is better than the more common specifications of the Usability requirement that we often see, such as: "2.4. The product will be more user-friendly, using Windows".

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 else has determined are valued. Deciding what sets of requirements are valuable is a Product Owner (Scrum) or Marketing Management function. Certainly only the IT-related value should be determined by the IT staff.

2.3. Define a 'Requirement' as a 'Stakeholder-Valued End State'

Do we all have a shared notion of what a 'requirement' is? I am afraid that another 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. Below I provide you with my latest 'opinion' about the best definition of 'requirement', but note it is a 'work in progress' and possibly not my final definition. Perhaps some of you can help improve this definition even further.

To emphasize 'the point' of IT systems engineering, I have decided to define a requirement as a "stakeholder-valued end state". You possibly will not accept, or use this definition yet, but this is the definition that I shall use in this paper, and I will argue the case for it. In addition, I have also identified, and defined a large number of requirement concepts [1]. A sample of these concepts is given in **Figure 5**.

Further, note that I make a distinction amongst:

1) A requirement (a stakeholder-valued end state)

2) A requirement specification

3) An implemented requirement

4) A design in partial, or full service, of implementing a requirement.

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 a defined [User] can successfully complete the 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.
Value [Market =USA, User = Seniors, Product = New Version, Task = Photo Tasks Set, Time Period = 2012]**:** 2 M USD.
Tolerable [Market = Asia, User = {Teenager, Early Adopters}, Product = Our New Version, Deadline = 2013]**:** 97% ± 3% < - Marketing Director Speech.
Fail [Market = Finland, User = {Android Mobile Phone, Teenagers}, Task = Phone + SMS Task Set, Product Release 9.0]**:** Less Than 95%.
Value [Market = Finland, User = {Android Mobile Phone, Teenagers}, Task = Phone + SMS Task Set, Time Period = 2013]**:** 30K USD.

Figure 4. A practical made-up Planguage example, designed to display ways of making the value of a requirement clear.

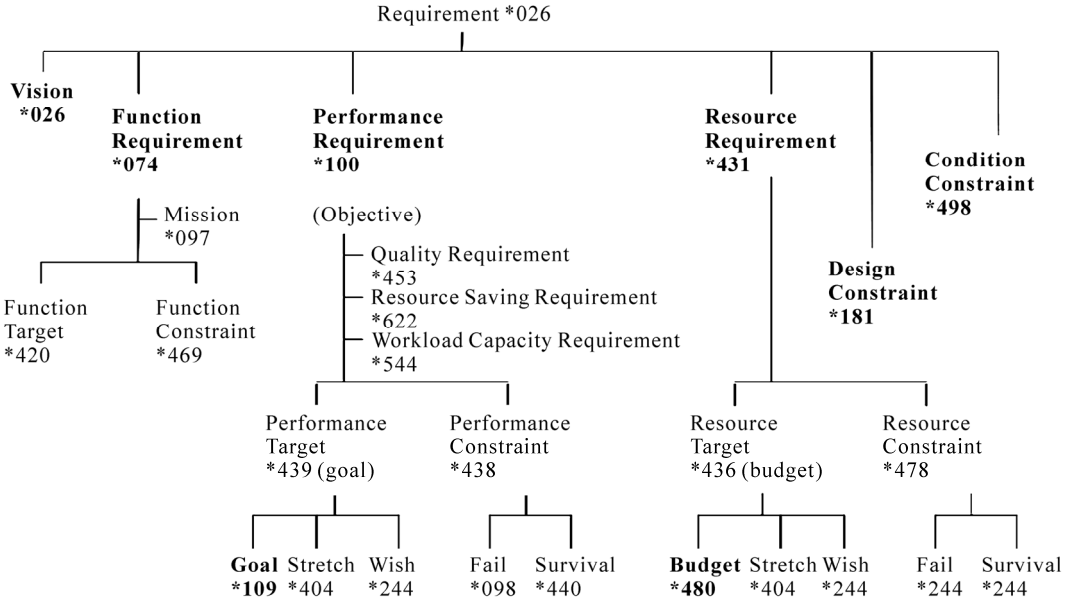

Figure 5. Example of Planguage requirements concepts.

These distinctions will be described in more detail later in this paper.

2.4. Think Stakeholders: Not Just Users and Customers!

Too many requirements specifications limit their scope to being too narrowly focused on user or customer needs. The broader area of stakeholder needs and values should be considered, where a 'stakeholder' is anyone or anything that has an interest in the system [1]. It is not just the end-users and customers that must be considered: IT development, IT maintenance, senior management, government, and other stakeholders matter as well.

2.5. Quantify Requirements as a Basis for Software Engineering

Some systems 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 as described by engineering professors, like Koen [2]. Instead these developers all too often produce requirements specifications consisting merely of words. No numbers, just nice sounding words; good enough to fool managers into spending millions for nothing (for example, "high usability").

Engineering is a practical bag of tricks. My dad was a real engineer (with over 100 patents to his name!), and I don't remember him using just words. He seemed forever to be working 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.

My life changed professionally, when, in my twenties, I read the following words of 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" [3]. Alternatively, more simply, also credited to 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). However, we seem to almost totally avoid the practice of quantifying these qualities, in order to make them clearly understood, and also to lay the basis for measuring and tracking our progress in improvement towards meeting our quality level requirements.

This art of quantification of any quality requirement should be taught as a fundamental to university students of software and management disciplines (as it is in other sciences and engineering). One problem seems to be that the teachers of software disciplines do not appreciate that quality has numeric dimensions and so cannot teach it. Note the problem is not that managers and software people cannot and do not quantify at all. They do. It is the lack of 'quantification of the qualitative'—the lack of numeric quality requirements—that is the specific problem.

Perhaps we need an agreed definition of 'quality' and '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, even 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 a 'scalar' attribute, since there are degrees of 'how well'. In addition to quality, there are other requirement-related concepts, such as workload capacity (how much performance), cost (how much resource), function (what we do), and design (how we might do function well, at a given cost) [1,4]. Some of these concepts are scalar and some, binary. See **Figures 6** and **7** for some examples of quality concepts and how quality can be related to the function, resources and design concepts.

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. One simple way to explore this is to search the internet. For example: "Intuitiveness scale measure" turns up 3 million hits, including this excellent study [5] by Yanga *et al*.

Several major corporations have top-level policy to quantify all quality requirements (sometimes suggested by me, sometimes just because they are good engineers). They include IBM, HP, Ericsson and Intel [1,4].

The key idea for quantification is to define, or reuse a definition, of a scale of measure. For example: (earlier given with more detail)

To give some explanation of the key quantification features in **Figure 8**:

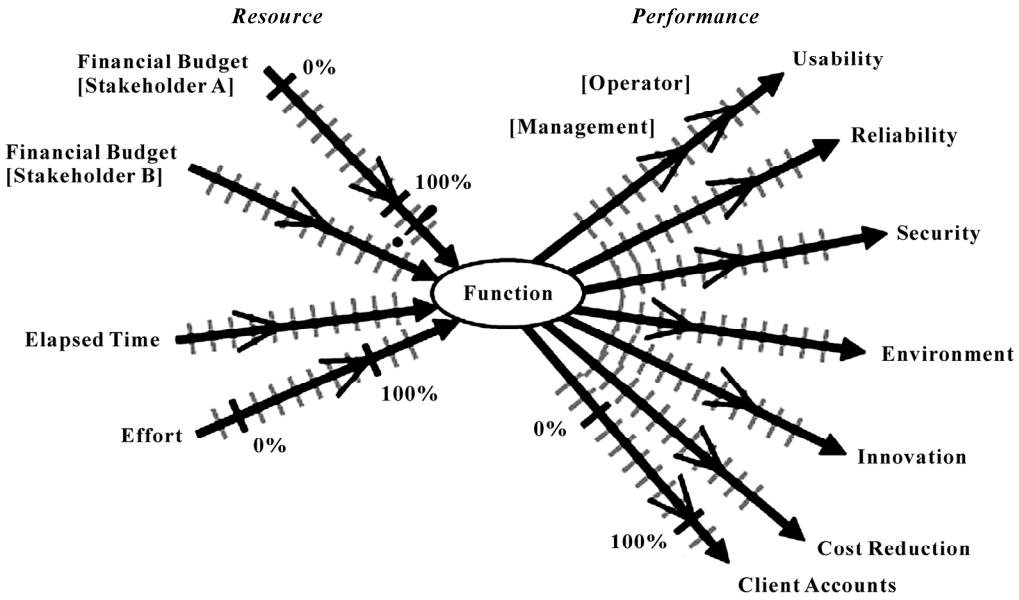

Figure 6. A way of visualizing qualities in relation to function and cost. Qualities and costs are scalar variables, so we can define scales of measure in order to discuss them numerically. The arrows on the scale arrows represent interesting points, such as the requirement levels. The requirement is not 'security' as such, but a defined, and testable degree of security [1].

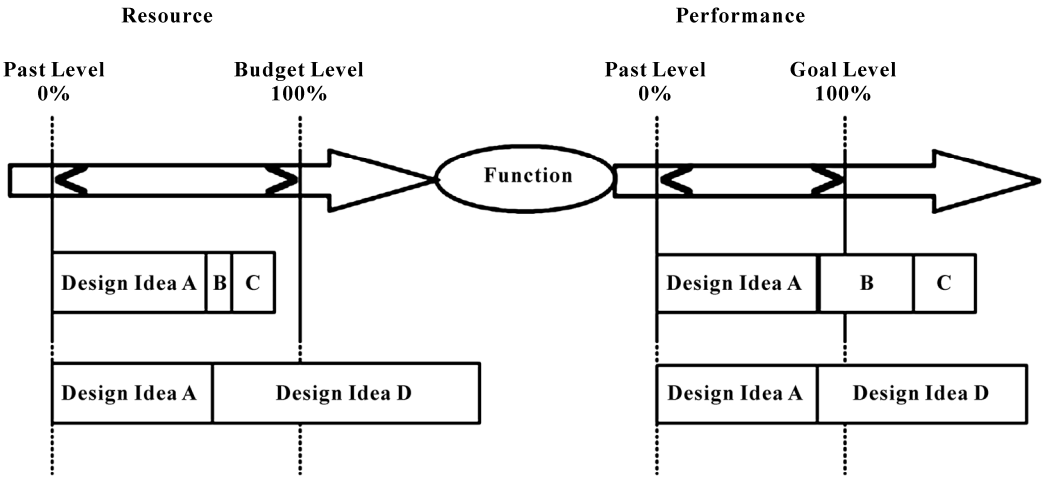

Figure 7. A graphical way of understanding performance attributes (which include all qualities) in relation to function, design and resources. Design ideas cost some resources, and design ideas deliver performance for given functions. Source [1].

1) Ambition is a high level 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.

2) 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 the reuse of the Scale, as a sort of 'pattern'.

3) 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, for example: Km/hour is not a speedometer, and a volt is not a voltmeter.

4) Goal is one of many possible requirement levels (see earlier detail for some others; Fail, Tolerable, Stretch, Wish, are other requirement levels). We are de-

fining a stakeholder valued future state (state = 80% ± 10%).

One *stakeholder* is 'USA Seniors'. The *future* is 2012. The requirement level type, Goal is defined as a very high priority, budgeted promise of delivery. It is of higher priority than a Stretch or Wish level. Note 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*!

Here is another example of quantification. It is the initial stage of the rewrite of Robustness from the **Figure 2** example. First we determined that Robustness is complex and composed of many different attributes, such as Testability. See **Figure 9**.

And see **Figure 10**, which quantitatively defines one of the attributes of Robustness, Testability.

Note this example shows the notion of there being different levels of requirements. Principle 1 also has relevance here as it is concerned with top-level objectives (requirements). The different levels that can be identified include: corporate requirements, the top-level critical few project or product requirements, system requirements and software requirements. We need to clearly document the

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.

Figure 8. A simple example of quantifying a quality requirement, 'Intuitiveness'.

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

Figure 9. Definition of a complex quality requirement, Robustness.

Testability:
Type: Software Quality Requirement.
Version: Oct 20, 2006.
Status: 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 of Testing] by a defined [Skill Level] of system operator under defined [Operating Conditions].
Goal [All Customer Use, Volume = 1,000,000 data items, Type of Testing = WireXXXX vs. DXX, Skill Level = First Time Novice, Operating Conditions = Field]**:** < 10 minutes.
Design: 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 HFS.

Figure 10. Quantitative definition of testability, an attribute of Robustness.

level and the interactions amongst these requirements.

An additional notion is that of 'sets of requirements'. Any given stakeholder is likely to have a set of requirements rather than just an isolated single requirement. In fact, achieving value could depend on meeting an entire set of requirements.

2.6. 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 and/or architecture, instead of what we really value—our real requirement [6]. There are explanatory reasons for this—for example solutions are more concrete, and what we want (qualities) are more abstract for us (because we have not yet learned to make them measurable and concrete).

The problems occur when we do confuse them: if we do specify the means, and not our true ends. As the saying goes: "Be careful what you ask for, you might just get it" (unknown source). The problems include:

1) You might not get what you *really* want,

2) The solution you have specified might *cost too much* or have bad *side effects*, even if you do get what you want,

3) There may be much *better solutions* you don't know about yet.

So how to we find the 'right requirement', the 'real requirement' [6] that is being 'masked' by the solution? *Assume* that there probably is a better formulation, which is a more accurate expression of our real values and needs. Search for it by asking 'Why?' Why do I want X, it is 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. This is a slight variation on the '5 Whys' technique [7], which is normally used to identify root causes of problems (rather than high level objectives).

Assume that our stakeholders will *usually* state their values in terms of some perceived means to get what they really value. Help them to identify (The 5 Whys?) and to acknowledge what they really want, and make that the 'official' requirement. Don't insult them by telling them that 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 '5 Whys?' process with you. See **Figure 11**.

Note that this separation of designs from the requirements does not mean that you ignore the solutions/designs/architecture when software engineering. It is just that you must separate your requirements including any mandatory means, from any optional means.

2.7. Focus on the Required System Quality, Not Just its Functionality

Far too much attention is paid to what the system must do (function) and far too little attention is given 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. See **Table 1**, which is from the Confirmit case study [8]. Here focusing on the quality requirements, rather than the functions, achieved a great deal!

2.8. Ensure there is 'Rich Specification': Requirement Specifications need Far More Information than the Requirement itself

Far too much emphasis is often placed on the requirement itself; and far too little concurrent information is gathered about its background, for example: who wants this requirement and why? What benefits do they perceive from this requirement? I think the requirement itself might be less than 10% of a complete requirement specification that includes the background information.

I believe that background specification is absolutely

Why do you require a 'password'? For Security!
What kind of security do you want? Against stolen information
What level of strength of security against stolen information are you willing to pay for? At least a 99% chance that hackers cannot break in within 1 hour of trying! Whatever that level 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!
The aim being that whatever technology we choose, it gets you the 99%?
Sure, thanks for helping me articulate that!

Figure 11. Example of the requirement, not the design feature, being the real requirement.

Table 1. Extract from confirmit case study [8].

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 market research report	65 min	20 min
Usability. Productivity: Time to grant a set of end-users access to a report set and distribute report login information	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 a response time < 500ms given a defined [Survey Complexity] and a defined [Server Configuration, Typical]	250 users	6000

mandatory: it should be a corporate standard to specify a great deal of this related information, and ensure it is intimately and immediately tied into the requirement specification itself.

Such background information is the part of a specification, which is *useful* related information, but is not *central* (*core)* to the implementation, and nor is it commentary. The central information includes: Scale, Meter, Goal, Definition and Constraint. Commentary is any detail that probably will not have any economic, quality or effort consequences if it is incorrect, for example, notes and comments.

Background specification includes: benchmarks {Past, Record, Trend}, Owner, Version, Stakeholders, Gist (brief description), Ambition, Impacts, and Supports. The rationale for background information is as follows:

1) To help judge value of the requirement
2) To help prioritize the requirement
3) To help understand risks with the requirement
4) To help present the requirement in more or less detail for various audiences and different 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.

See **Figure 12** for an example, which illustrates the help given by background information regarding 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 costs to reach this level might be excessive.
Design Suggestion: Triple redundant software and database system.
Goal [Shops]: 30,000 hours < - 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 for the component parts that they might no longer require.
Design Suggestion: Simplification and reuse of known components.

Figure 12. A requirement specification can be embellished with many background specifications that will help us to understand risks associated with one or more elements of the requirement specification [9].

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. See **Figure 13** for a requirement template for function specification [1], which hints at the richness possible

TEMPLATE FOR FUNCTION SPECIFICATION <with hints>

Tag: <Tag name for the function>.

Type: <{Function Specification, Function (Target) Requirement, 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 someplace 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>.

Figure 13. A template for function specification [1].

for background information.

2.9. Carry out Specification Quality Control (SQC)

There is far too little quality control of requirements, against relevant standards for requirements. All requirements specifications ought to pass their quality control checks before they are released for use by the next processes. Initial quality control of requirements specification, where there has been no previous use of specification quality control (SQC) (also known as Inspection), using three simple quality-checking rules ('unambiguous to readers', 'testable' and 'no optional designs present'), typically identifies 80 to 200+ words per 300 words of requirement text as ambiguous or unclear to intended readers [10]!

2.10. Recognise That Requirements Change: Use Feedback and Update Requirements as Necessary

Requirements must be developed based on on-going feedback from stakeholders, as to their real value. Stakeholders can give feedback about their perception of value, based on *realities*. The whole process is a 'Plan Do Study Act' cyclical learning process involving many 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: for example, if they are committed to—in contracts and fixed specifications.

3. Who or What will Change Things?

Everybody talks about requirements, but few people seem to be making progress to enhance the quality of their specifications and improve support for software engineering. I am pessimistic. Yes, there are 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 organizations building software. The vast majority of IT systems development teams we encounter are not highly motivated to learn or practice first class requirements (or anything else!). Neither the managers nor the developers seem strongly motivated to improve. The reason is that they get by with, and get well paid for, failed projects.

The universities certainly do not train IT/computer science students well in requirements, and the business schools also certainly do not train managers about such matters [11]. The fashion now seems to be to learn over-simplified methods, and/or 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. So, it is the only the elite few organizations and individuals who do in fact realize the competitive edge they get with better practices [8,12]. 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 perhaps, denied. Perhaps 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 of CMMI and the various organizations at Level 5).

Excuse my pessimism! I am sitting here writing with the BP Gulf Oil Leak Disaster in mind. The BP CEO Hayward just got his reward today of £11 million in pension rights for managing the oil spill and 11 deaths. In 2007, he said his main job was "to focus 'laser like' on safety and reliability" [13]. Now how would you define, measure and track those requirements?

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

4. Summary

Current typical requirements specification practice is woefully inadequate for today's critical and complex systems. There seems to be wide agreement about that. 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!

We know what to do to improve requirements specification, 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 specify requirements to a decent standard? If there are some executives, governments, professors and/or consultancies, who want to try to improve their project requirements, then I suggest start by seeing how your current requirements specifications measure up to addressing the ten key principles in this paper.

5. Acknowledgements

Thanks to Lindsey Brodie for editing this paper.

REFERENCES

[1] T. Gilb, "Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage," Elsevier Butterworth-Heinemann, Boston, 2005.

[2] B. V. Koen, "Discussion of the Method: Conducting the Engineer's Approach to Problem Solving," Oxford University Press, Oxford, 2003.

[3] L. Kelvin, "Electrical Units of Measurement," a Lecture Given on 3 May 1883, Published in the Book "Popular Lectures and Addresses, Volume 1," 1891.

[4] T. Gilb, "Principles of Software Engineering Management," Addison-Wesley, Boston, 1988.

[5] Z. Yanga, S. Caib, Z. Zhouc and N. Zhoua, "Development and Validation of an Instrument to Measure User Perceived Service Quality of Information Presenting Web Portals," *Information & Management*, Vol. 42, No. 4, 2005, pp. 575-589.

[6] T. Gilb, "Real Requirements". http://www.gilb.com/tiki-download_file.php?fileId =28

[7] T. Ohno, "Toyota Production System: Beyond Large-Scale Production," Productivity Press, New York, 1988.

[8] T. Johansen and T. Gilb, "From Waterfall to Evolutionary Development (Evo): How we Created Faster, More User-Friendly, More Productive Software Products for a Multi-National Market," *Proceedings of INCOSE*, Rochester, 2005. http://www.gilb.com/tiki-download_file.php?fileId=32

[9] T. Gilb, "Rich Requirement Specs: The Use of Planguage to Clarify Requirements," http://www.gilb.com/tiki-download_file.php?fileId=44

[10] T. Gilb, "Agile Specification Quality Control, Testing Experience," March 2009. www.testingexperience.com/testingexperience01_08.pdf

[11] K. Hopper and W. Hopper, "The Puritan Gift," I. B. Taurus and Co. Ltd., London, 2007.

[12] "Top Level Objectives: A Slide Collection of Case Studies". http://www.gilb.com/tiki-download_file.php?fileId=180

[13] "Profile: BP's Tony Hayward, BBC Website: News US and Canada," 27 July 2010. http://www.bbc.co.uk/news/world-us-canada-10754710

