Rich Requirement Specs:
The use of Planguage to clarify requirements.

Iver Holtersvei 2, NO-1410 Kolbotn, Norway, Tom@Gilb.com,
www.Gilb.com, +47 66801697

Abstract:
I believe that most requirements specifications in practice are very poor in clarity, and in content. I believe that in addition to tackling the clarity problem by a variety of rich specification devices, we need to make a requirement specification 'work harder' to satisfy a large number of needs by adding 'background' to the requirement. The needs I am referring to include: prioritization, risk management, change management, presentation, justification, testability, integration, quality control, and other purposes. To do this I have, over the years developed a requirement specification language, as a subset of my Planning Language (Planguage). This is has been developed by practical need in international industry over decades, and supplemented with some recent ideas. The more detailed version of the Requirements Language is documented in my book Competitive Engineering, which is a handbook defining concepts rigorously. This paper will give an overview of the conceptual basis and some detail as a sample. By 'rich' I mean substantially more detail for each requirement than is usual. By 'background' I mean information related to the requirement that is not the requirement itself.

I will present the Requirements Language in terms of some basic principles:

Principle R1: The requirement should be a reusable object.

A requirement needs to be referenced by a large number of instances in the totality of project documentation; tests, design, architecture, risk analysis, quality control – for example. Each requirement must be specified once, one single master version should exist, and all uses of the requirement must refer to that master version. They should in particular not copy and paste an instance of it, or rewrite it in subtly different versions.

This lays the basis for an investment in the master requirement. It is worth doing properly, to a high level of quality, clarity and sophistication. You only have to do it once, for the master, and then all reuse of that master requirement benefits.

Here are some of the parameters that are useful to fulfill this principle:

Reliability:
Scale: Mean Time Between Failure.
Goal: 20,000 hours.

This is a simple starting point requirement. The only thing that helps make it reusable here is a unique tag (Reliability). This tag cannot be used by any other requirement in the

project (without some qualification to make it unique). The Scale of measure and the Goal level specifications are *core* requirement specification.

Now we can add some additional optional parameters (Scale and Goal were parameters in the requirement language, in the example above).

Reliability:
Version: November 16, 2003
Owner: Quality Director
Author: John Engineer
Stakeholders: {Users, Shops, Repair Centers}.
Status: Approved for Design
Scale: Mean Time Between Failure.
Goal: 20,000 hours.

The **added specifications** in this example are not core requirement specification. They are background specification. They add something to the core specification. Now notice that many of these types of things are traditionally done at the level of a requirements document, for a set of 'all' requirements for a project. The fact that we are moving them to the level of a single requirement implies:

- we are treating the requirement as a distinct 'object'.
- we intend to update this requirement independently of others
- we intend to consult (owner, stakeholders) about changes to this requirement alone.
- we intend to quality control this requirement independently of others
- there can be many individual authors, each writing their individual requirements, but each clearly accountable for their 'baby'. No hiding in the committee.

The requirement is reusable in the sense of reusable on this project. There are aspects of the Requirements Language that make certain aspects of a requirement reusable across projects. For example:

Usability:
Scale: The probability in % that defined **Users** can successfully complete defined **Tasks** under defined **Conditions**.
Goal [Users = Novices, Tasks = Most Difficult, Conditions = {Noise, Pressure}] 60%.
Goal [Users = Experts, Tasks = Average, Conditions = {Noise, Movement}] 80%.

In this case the scale of measure definition has three scale qualifiers (Users, Tasks, Conditions). These are defined by corresponding scale variables in the Goal statement. This allows reuse of the scale in the same requirement definition, but it also enhances the capability for reuse of that scale definition in entirely different projects.

Principle R2: The requirement should give information allowing us to check its quality.

What is the ‘quality’ of a requirement specification? Different people are going to give very different answers. But here is mine. There are two distinct quality aspects of a requirement specification:
• the *intelligibility* of the specification (even though it is wrong or bad!)
• the *relevance* of the specification to the stakeholder world (does it really fulfill the real needs of stakeholders?) – even though it may be unclear
The requirements language has a large array of tools addressing these two aspects.

Let us first look at the issue of *intelligibility*:

Reliability:
Scale: Mean Time Between Failure.
Goal: 20,000 hours.

Some simple notions of intelligibility in this basic example are:
• the requirement is given an intelligible reference name (Reliability)
• the requirement is described by a set of clear language parameters (Scale, Goal)
• distinct concepts are clearly separated (the name, the scale of measure, the target level).

The language parameters themselves are formally defined in the requirement language concept glossary, so that the parameters have an exact meaning. For example:

Goal **Concept *109 March 15, 2003**

A goal is a primary numeric target level of performance.

A Goal parameter is used to specify a *performance* target for a scalar attribute.

A Goal level is specified on a defined scale of measure with its relevant qualifying conditions [time, place, event].

An implication of a Goal specification is that there is, or will be, a *commitment* to deliver the Goal level (something *not* true of a Stretch or Wish target specification).

Any commitment is based on a trade-off process, against other targets, and considering any constraints. The specified goal level may need to go through a series of changes, as circumstances alter and are taken into consideration.

A specified Goal level will *reasonably satisfy* stakeholders. Going beyond the goal, at the cost of additional resources, is not considered necessary or profitable – even though it may have *some* value to do so.

(and this snippet does not include the extensive notes also found in the glossary)

Of course the exact *content* itself is critical to intelligibility, but we are just mentioning some of the intelligibility concepts, not all of them.

Now let us look at the issue of *relevance*.

What if the requirement as stated represents at best only part of the constituency. One stakeholder, but not all. What if the requirement does not meet the needs of the main authority behind the need? What if the requirement is a misinterpretation of a stated stakeholder need? These and many similar considerations doom even a clear requirement to fail the test of relevance. Here is how the requirements language can help us discoved such problems.

Reliability:
Owner: Quality Director
Status: Approved by Owner 17 November 2003.
Version: 17 December 2003.
Author: John Engineer
Stakeholders: {Users, Shops, Repair Centers}.
Scale: Mean Time Between Failure.
Goal: 20,000 hours.

Some of these parameters we met earlier, but now we can point out additional purposes, for relevance testing. When we know the owner of a requirement is the quality director, who last approved a change 17 November, but we can see there is a new version later, we know we need to get the requirement owner to approve the change. If there is some question about the content we know that the first point of checking it out is with the author, who may admit it can be improved, or that they failed to realize something, If we wanted to check the relevance of the scale or the goal, we know we can analyze or speak to the listed stakeholders.

Reliability:
Owner: Quality Director
Author: John Engineer
Stakeholders: {Users, Shops, Repair Centers}.
Scale: Mean Time Between Failure.
Goal[Users]: 20,000 hours**. <- Customer Survey, 2004**
Goal[Shops]: 30,000 hours. **<- Dixons Chain [Quality Director].**

Most details should have an accompanying ‘source’ annotation; done here by a ‘<-‘ source arrow symbol. The source information is primarily used in quality control and change processes, in order to know exactly where to check the specification. If you take a look at most people’s specs you will find that source information at the detailed level is the exception, rather than the rule it should be. The source information indirectly tells us about levels of authority, and thus can be used to understand the priority of a requirement level.

Principle R3: The requirement should explicitly document its relationships to designs, tests, stakeholders, sources, use cases and all other interesting things

Reliability:
Owner: Quality Director
Author: John Engineer
Stakeholders: {Users, Shops, Repair Centers}.
Scale: Mean Time Between Failure.
Meter [Operations] Automated Log.
Goal[Users]: 20,000 hours**.** <- Customer Survey, 2004
Test [Integration Testing]: 60,000 hours continuous minimum.
Goal[Shops]: 30,000 hours. <- Dixons Chain [Quality Director].
Test [Field Trials]: defined by Customers in their Contracts, Default: MTBF Test.
Design Proposal: Distinct Triple Voting Software <- JJ Architect.

A Meter specified one or more ways to measure along the defined scale, what the level of performance is, for example in operation. Tests can be specified, in detail, schematically or by reference to detailed test plans and test cases. A design proposal is not a firm design commitment, just a note documenting some early design ideas; maybe indicating that ambitious target levels have a practical technical solution.

Principle R4: The requirement should be a future state, and not unintentionally a design to get there.

Usability:
Scale: The probability in % that defined **Users** can successfully complete defined **Tasks** under defined **Conditions**.
Goal [Users = Novices, Tasks = Most Difficult, Conditions = {Noise, Pressure}] 60%.
Goal [Users = Experts, Tasks = Average, Conditions = {Noise, Movement}] 80%.
Stretch [Users = Experts, Tasks = Average, Conditions = {Noise, Movement}] 85%.
Wish [Users = Experts, Tasks = Average, Conditions = {Noise, Movement}] 90%.
Ideal [Users = Experts, Tasks = Average, Conditions = {Noise, Movement}] 100%.

In addition to the Goal statement, the one requirement level we take as a serious commitment, there are three other target notions.

Stretch represents a level somewhat above the corresponding qualifiers Goal level. It has added stakeholder value, but we do not believe we have either the resources or the technology to commit to the Stretch level. But we document, as background, not as a 'required' level, that there is stakeholder value in reaching that level. Maybe later we can find additional resources, or better technology and reach the stretch level. It is a sort of motivator for engineers to do that little extra. But you could say it is a requirement level that is not required.

Wish-level specification represents a target level of performance that has some value for some stakeholder. Wish specs represents a stakeholder 'need' – that perhaps can later be converted is not a Goal level. But first we would need to find a suitable design to get to that Wish level, then we would need to estimate costs for using the design, and then finally we would need to accept that cost within our total budget. Then we could convert the Wish level to a Goal level. You could say that the non-committed Wish

specification, can be viewed as a way of 'begging for resources'. We won't promise to commit to the wish as a goal level, unless we get sufficient resources.

The Ideal level is rarely used in practice, but it is a useful concept. It represents perfection that all stakeholders, in principle, want. But they do not want to pay the infinite price that is normally the cost of perfection. The ideal, therefore is not a requirement, but it can serve as a conceptual reminder that we cannot just satisfy peoples impossible dreams.

Notice that the level of performance intentionally says nothing about the actual design that will be used to deliver that level of performance. That strict division between required performance levels and the next stage, the design process, is necessary to free the architect and the design engineer to find the best solutions to the problem posed by the complete set of requirements: all performance levels, all cost levels and all constraints. It would be logically wrong to decide on a particular design on the basis of a single dimension of performance. That would be sub-optimization.

Principle R5: Complex requirements should have a clear hierarchy, not a vague summary.

Many top level performance requirements seem to defy quantification. This is normally because they are not really a single scalar dimension. They need to be decomposed from being a 'complex' notion (consists of many dimensions) to a set of elementary dimensions. It is a simple as making a list of the aspects that we associate, and agree to associate with, a performance attribute. Then we can normally define a suitable scale of measure for the elementary aspects. Sometimes further decomposition is necessary.

Adaptability:
Scale:
time needed to [Adapt]
a defined [System]
from a defined [Initial State]
to another defined [Final State]
using defined [Means].

Example: this is the high level concept of adaptability, defined using a number of scale qualifiers (like [Adapt]). Another approach to defining adaptability is to treat it as a set of defined classes of adaptation (which is what the Adapt qualifier does too).

1.1.2.1 Portability:
Scale:
the cost to transport
a defined [System]
from a defined [Initial Environment]
to a [Target Environment]
using defined [Means].

1.1.2.2 Extendability:
Scale:
the cost to add
to a defined [System]
a defined [Extension Class]
and defined [Extension Quantity]
using a defined [Extension Means].
1.1.2.3 Improvability:
Scale:
the cost to add
to a defined [System]
a defined [Improvement] and
defined [Quantity]
using a defined [Means].

Example: here is a template for defining Adaptability as a set of three types of Adaptability. The tag 'Adaptability' is simply a way of referring to the three of these.

The same 'Adaptability' concept was also decomposed into quantified aspects like this:

Demonstrability
Customer self-demonstrability 7.1.1
Our professional demonstrability 7.1.2

Installability
Customer 7.2.1
Professional on-site 7.2.2
Professional ex-works 7.2.3

Interchangeability
Replaceability 7.3.1
Movability 7.3.2
Interface 7.3.3

Upgradability
Node addability 7.4.1
Connection addability 7.4.2
Application addability 7.4.3
Subscriber addability 7.4.4

Portability
Data portability 7.5.1
Logic portability 7.5.2
Command portability 7.5.3
Media portability 7.5.4

Connectability
(was detailed elsewhere)

Example of many different aspects of adaptability, used to define a major product line for a UK Manufacturer.

Principle R6: Requirements should be rich in information allowing us to determine their current priority.

The priority of a requirement should not be pre-determined (by a prioritized list or fixed weights). The information needed to make decisions at a given moment in time, about the requirement priority should be partly present, in the requirements specification. Partly it needs to be held in both design specifications (what do the designs cost to support a requirement), in project progress information (how much time, money, effort resources remain to support given requirements and their design).
This will allow you to make smarter decisions, based on better and more up-to-date information than otherwise.

Usability:
Scale: The probability in % that defined Users can successfully complete defined Tasks under defined Conditions.

======== Constraint Level Requirements ================
C1: Fail Level Users = Novices, Tasks = Most Difficult, Conditions = {Noise, Pressure}, **Release 1.0**] 40%. **<- Marketing Director.**
C2: Survival Level Users = Novices, Tasks = Most Difficult, Conditions = {Noise, Pressure}, **Release 1.0**] 20%.
Authority: **Marketing Director.**

======== Target Level Requirements ================
T1: Goal [Users = Novices, Tasks = Most Difficult, Conditions = {Noise, Pressure}, **Release 1.0**] 60%. **<- Marketing Director.**
T2:Goal [Users = Experts, Tasks = Average, Conditions = {Noise, Movement}, **Release 2.0**] 80%. **<- User Group Proposal.**
T3: Stretch [Users = Experts, Tasks = Average, Conditions = {Noise, Movement}] 85%.
T4: Wish [Users = Experts, Tasks = Average, Conditions = {Noise, Movement}] 90%.

Example: Many elements of a requirement specification are able to give clues as to the relative priority the requirement has. This example is a small sample of the possibilities, but it will give an idea of the concept of there being many specification elements that are sources of understanding relative priorities.

In the example above:
• *The two Goal target levels have a higher priority than the Stretch level. The Stretch levels, in general (but it does depend on the [qualifiers] being otherwise equal) have a higher priority than the Wish level.*
• *The two Constraint levels (Fail, Survival) – qualifiers being equal – have higher priority than any equivalent target level (T1 to T4).*

• *An earlier release date (Release 1.0) , at that date, has a higher priority than a later release date.*
• *When the source (<-) or authority for a specified level has more power or clout than another source or authority, then that can partly determine the priority of the requirement.*

Principle R7: Performance and cost requirements should contain a rich specification of targets, constraints, and time/space/event qualifiers.

Usability:
Scale: The probability in % that defined Users can successfully complete defined Tasks under defined Conditions.

======== Constraint Level Requirements ================
C1: Fail Level Users = Novices, Tasks = Most Difficult, Conditions = {Noise, Pressure}, Release 1.0] 40%. <- Marketing Director.
C2: Survival Level Users = Novices, Tasks = Most Difficult, Conditions = {Noise, Pressure}, Release 1.0] 20%.
Authority: Marketing Director.

======== Target Level Requirements ================
T1: Goal [Users = Novices, Tasks = Most Difficult, Conditions = {Noise, Pressure}, Release 1.0, **IF Competitor New Release Available**] 60%. <- Marketing Director.
T2:Goal [Users = Experts, Tasks = Average, Conditions = {Noise, Movement}, Release 2.0] 80%. <- User Group Proposal.
T3: Stretch [Users = Experts, Tasks = Average, Conditions = {Noise, Movement}, **IF Competitors >80%]** 85%.
T4: Wish [Users = Experts, Tasks = Average, Conditions = {Noise, Movement}] 90%.

Using almost the same example, this example carries out the idea of the principle. There are six different requirement levels. Each one applies to a different set of times, types of users, types of tasks and event conditions. All six constraint and target requirement levels are distinctly different requirements; but they have a common performance dimensions they are helping to manage (Usability).

Principle R8: Any class of requirement should be specified, and the classes should be well defined.

A requirement type can generally be deduced from it's content, without explicit specification. But there are many different types of requirement, and an explicit specification is a useful summary and a useful control of expectations about the specification structure and content. For example Type: Performance expects Scalar

specification, but Type: Design Constraint expects design definition and perhaps information about expected impacts on cost and performance levels.

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
Goal [Shops]: 30,000 hours. <- Dixons' Chain [Quality Director].

Example: explicit type of specification declaration, using 'Type' parameter.

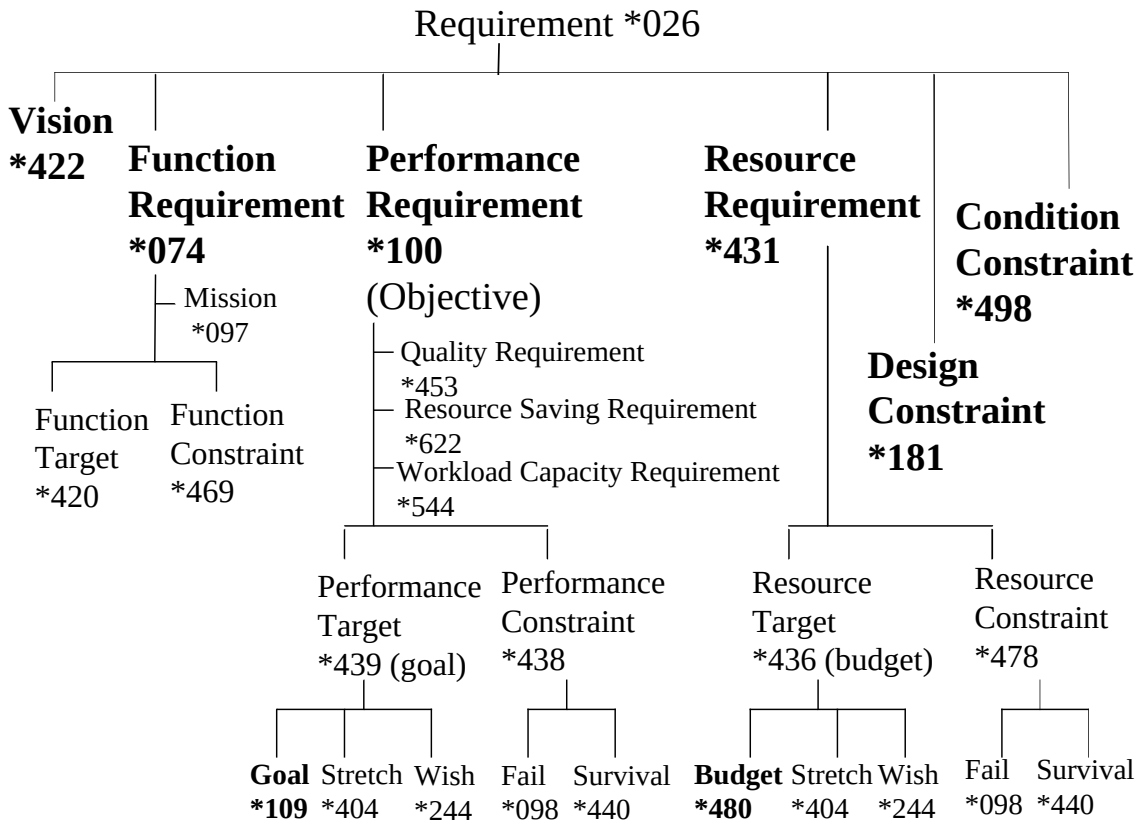

Figure: the structure of basic requirement types.

In addition to clearly separating different requirement specifications, the 'Type ' specification is also used to differentiate non-requirement specifications such as designs and project plans. The Type specification is also part of making individual specifications into freestanding 'objects'. Different specification types can be freely mixed without the necessity of segregating them into different document types. They are more readily retrieved and analyzed as elements of a specification database.

Principle R9: The requirement should be rich in information that allows us to understand, spot, analyze, present, and mitigate risks of deviation from that requirement.

For example:

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.

Principle 10: The requirement should be easy to specify without special tools, and it should be easy for any enterprise to extend or modify the requirement language to their own purposes.

The Planning Language is designed to be written using any tools, whiteboard, paper, MS Word, Excel, PowerPoint, or more sophisticated requirements software. The language is freely modifiable by any user for any purpose of tailoring or specialization. There are no costs or permissions necessary.

Author Bio

Tom has been an independent consultant, teacher and author, since 1960. He mainly works with multinational clients; helping improve their organizations, and their systems engineering methods.
Tom's latest book is 'Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage' (Summer 2005).
Other books are 'Software Inspection' (with Dorothy Graham, 1993), and 'Principles of Software Engineering Management' (1988). His 'Software Metrics' book (1976, OoP) has been cited as the initial foundation of what is now CMMI Level 4.
Tom's key interests include business metrics, evolutionary delivery, and further development of his planning language, 'Planguage'. He is a member of INCOSE and is an active member of the Norwegian chapter NORSEC. He participates in the INCOSE Requirements Working Group, and the Risk Management Group.

Email: Tom@Gilb.com
URL: http://www.Gilb.com

Version Nov 9 2005