Requirement Relationships: A Theory, some Principles, and a Practical Approach.

Drafted initially Sunday, 1 October 2006.

Introduction:

This paper will argue that the 'conventional ideas' [NASA 97, INCOSE01, Raytheon06] of how requirements relate to other entities is unnecessarily weak in relation to the complex demands placed on a systems engineering task. We will argue that it would improve systems engineering requirements practice if we were to invest substantially more in effort to determine, and to specify, at least a dozen or two useful relationships for *each* requirement. We will argue that the nature or variety of these relationships should but relatively unlimited (by a standard or tool), and should be whatever is useful to the engineering work. In addition, we need to keep the requirement relationship specifications, together with the *core* requirement itself, in a reusable requirement 'object' (a mini database about each discrete requirement, which is tool independent). Systematic rules and conventions, like those illustrated, will enable more-automated use (analysis, presentation, consistency checking, reuse) of requirements, even with simple text string searching.

The Theory:

A requirement is a *client* stakeholder *need* that a *server* stakeholder is *planning* to satisfy.

A *server stakeholder* attempts to deliver some results to satisfy the needs of a client stakeholder.

A *client stakeholder* states needs, approves requirements, and receives benefits or results produced by a server stakeholder.

Less formally, and less generally, we could say:

A **requirement** is a stakeholder need that a developer is planning to satisfy[1]

Another way to bring out the point about 'relative requirements' is to say that:
One person's means is another's ends.

*A **requirement relationship specification** is any explicit specification that connects a requirement to any other specification, event, condition, stakeholder or entity.*

[1] These 3 definitions were developed May 2005 after my book Competitive Engineering went to print. I view them as an important generalization compared to what I had earlier.

The implication of the requirement definitions above is that:

1. requirements are *only* requirements relative to defined stakeholders
2. requirements *imply* a decision to try to satisfy a stakeholder need
3. Normal requirements specifications should be specifications with a number of strongly implied relationships (like: needs, results, benefits, costs, stakeholders, degree of satisfaction, stakeholder priority, and many more).
4. There can be more than one stakeholder type that sponsors delivery of a single requirement.
5. A single requirement will be associated with at least one client stakeholder, and at least one server stakeholder – but probably more than one of each.
6. There can be more than one stakeholder type that is impacted (beneficially or negatively) from delivery of a requirement.
7. Specific instances of a stakeholder type (example different operational users of a system) may differ in their needs, and in their 'degree of satisfaction from delivery' of a given system requirement.

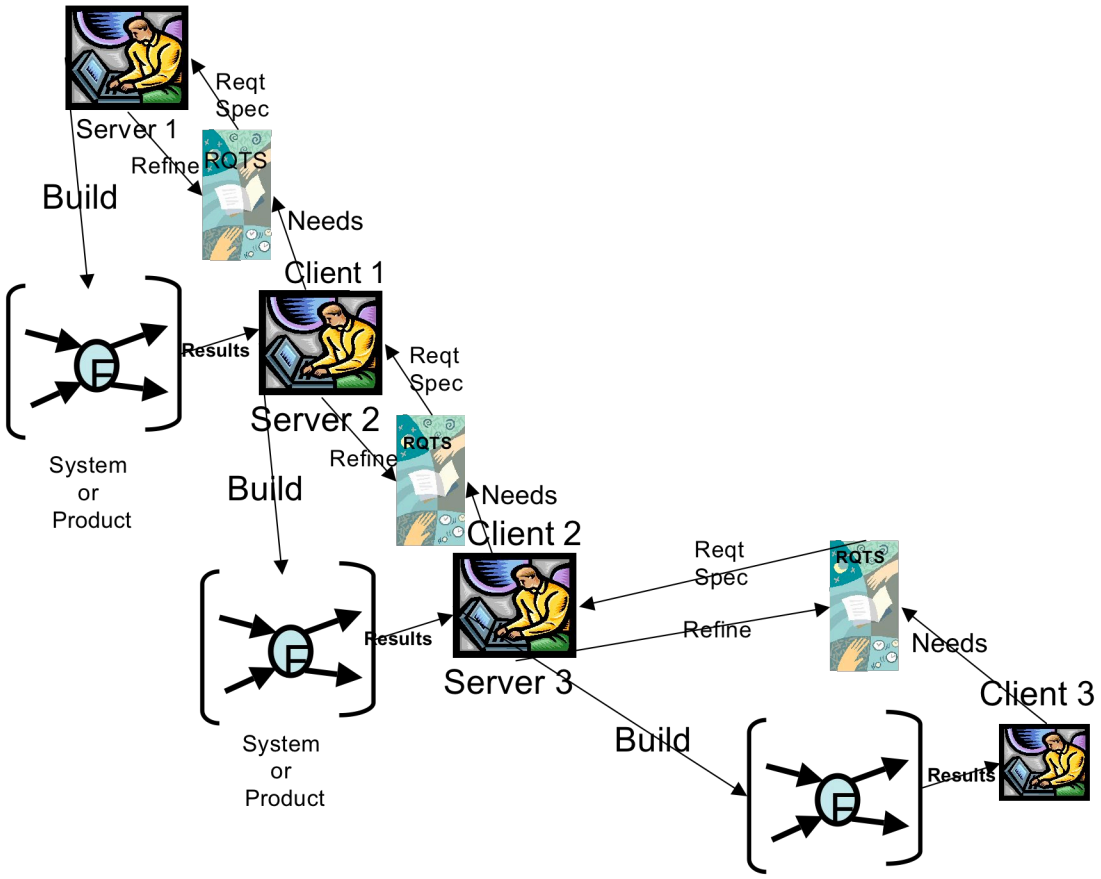


Figure The multiple levels of client stakeholders. The dual roles of stakeholders in having their needs satisfied by their 'servers', while themselves satisfying other (client) stakeholders.

The 'Requirement Relationship' Principles:

1. THE CLIENT STAKEHOLDER PRINCIPLE: A requirement specification that has no identified *client* stakeholder, is not a *valid* requirement . *Because - we cannot ascertain its usefulness or value to a given stakeholder.*
2. THE SERVER STAKEHOLDER PRINCIPLE: A requirement specification that has no specified, or implied, *server* stakeholder(s) is not yet *seriously* planned for real implementation. *Thus we cannot understand who will deliver it, when, or how efficiently*
3. THE REQUIREMENT RELATIONSHIPS PRINCIPLE: A single requirement can have *any useful number* and *types* of relationships that are worth specifying. *The total costs of specification should be less than the expected benefits in the* long term *for the system.*
4. THE EARLY RELATIONSHIP PRINCIPLE: Failure to deal with requirement relationships in the *requirement* specifications *themselves* will have the effect of increasing development and maintenance costs. *Because the relationships will then more likely be sensed, and dealt with, downstream, in design, testing and operation or even decomissioning.*
5. THE DYNAMIC RELATIONSHIP PRINCIPLE: Requirement Relationships are not static, nor are they are all *determinable* initially. *Consequently we need to track them as they emerge and change; we need to verify them, and we need to analyze the consequences of any change in requirement relationships.*
6. THE RELATIONSHIPS ARE 'CRITICAL KNOWLEDGE' PRINCIPLE: The requirement *relationship* knowledge is *itself* far more valuable and critical than the requirement alone. *This is because it potentially helps us to impact greater value and scope, earlier and better than we otherwise would be aware of, or would deal with. The requirement itself might change but most relationships might remain as useful facts*
7. THE 'REQUIREMENT REVIEW BASIS' PRINCIPLE: All requirement review processes are dependent on the quality and quantity of requirement relationship information available. *Otherwise we risk approving requirements in ignorance of critical facts.*
8. THE RISK MANAGEMENT PRINCIPLE: The Risk Management process is continuously dependent on the quality of requirement relationship information. *All requirement relationship specifications help us to identify and manage risks.*
9. THE 'BUTTERFLY EFFECT' PRINCIPLE: Even *one single* fault in a requirement relationship specification can be the root cause of project or system failure. *It is impossible to be sure that even a single missing or incorrect requirement relationship specification will be unable to severely or critically damage your engineering effort.*
10. THE DESIGN RELATIONSHIP PRINCIPLE: All *architecture and design* specifications must follow the same relationship specification principles, as their 'near cousins', requirements. *This is because, all 'solutions, means, designs, architectures, and strategies' are themselves also requirements, as viewed by other stakeholders.*

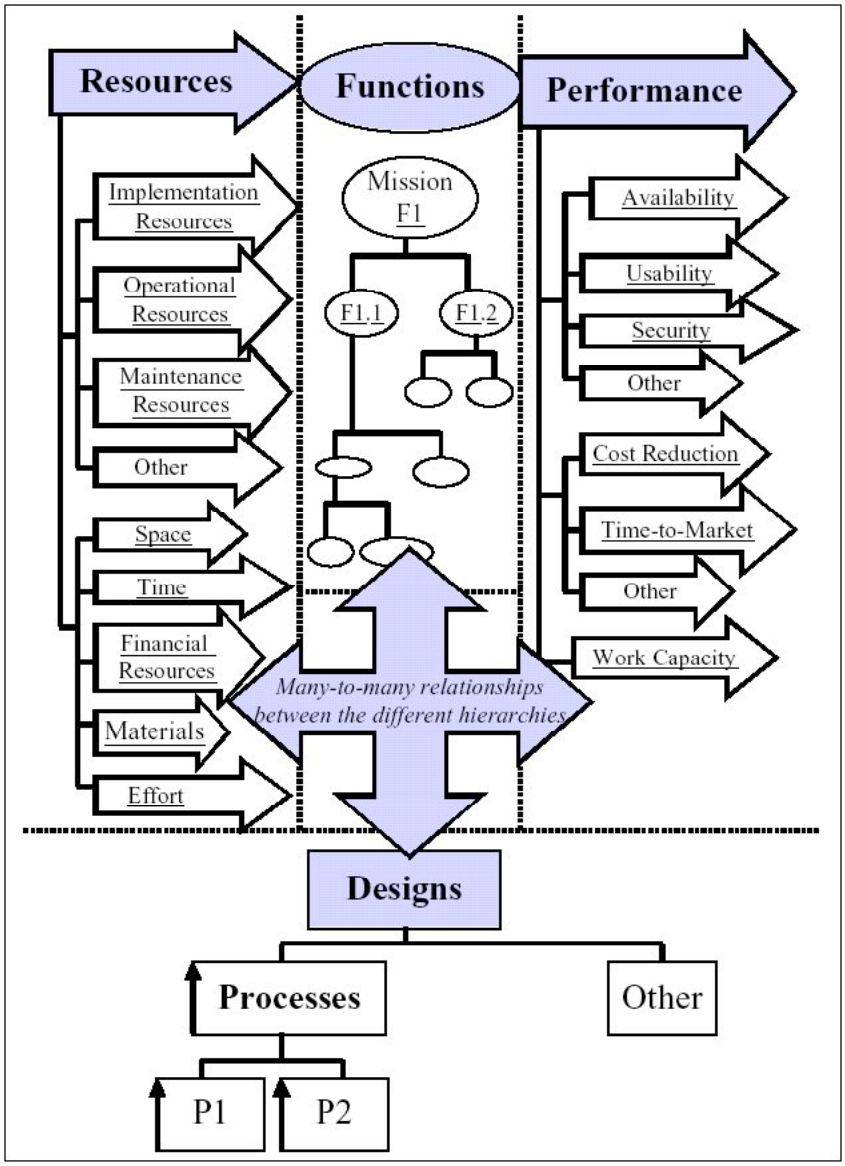


Illustration: This shows the four main system attribute types: resource, function, performance and design. It also shows the processes, which implement the functions. Using Planguage, the complex relationships amongst these four different types can be specified. For example, a specific performance level might apply only to a handful of functions; rather than the entire system. Or, a function might be implemented by several processes. Or, different resources can be specifically allocated to different functions. [source: CE 2005, Figure 3.3]:

The Practical Implementation:

Here are some practical examples of describing requirement relationships using the planning language 'Planguage'.

Classes of Requirement Relation Specification

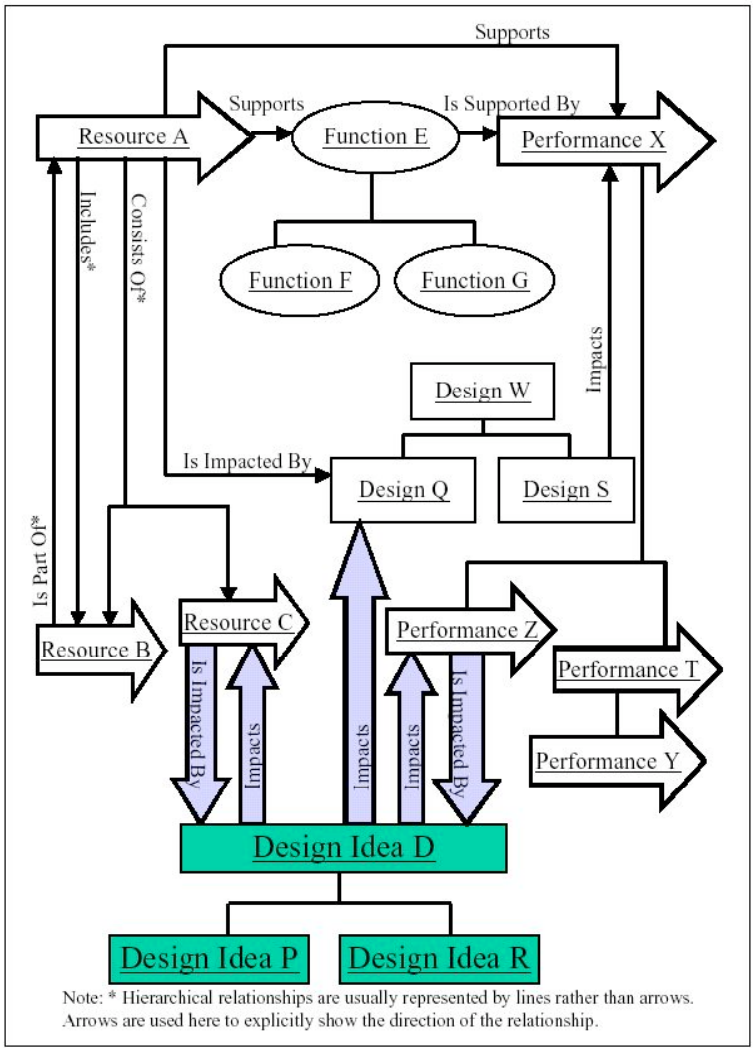


Diagram showing how to express the relationships amongst attributes, between attribute and design idea and, amongst design ideas.
The 'linking' terms include: Consists Of, Includes, Impacts, Is Impacted By, Supports, Is Supported By and, Is Part Of. Note: Not all potential types of relationships are shown. [Source: Gilb CE 2005, Figure 2.5]

Some Planguage parameters which define relationships between requirements and other system concepts:

- Authority
- Source
- Owner
- Author
- Implementer
- Impacts
- Supports

•Supported By
•Version
•Derived From
•Sub-component of
•Sub-components {list}
•Dependencies
•Contract
•Test Case
•Scenario
•Model
•And more!

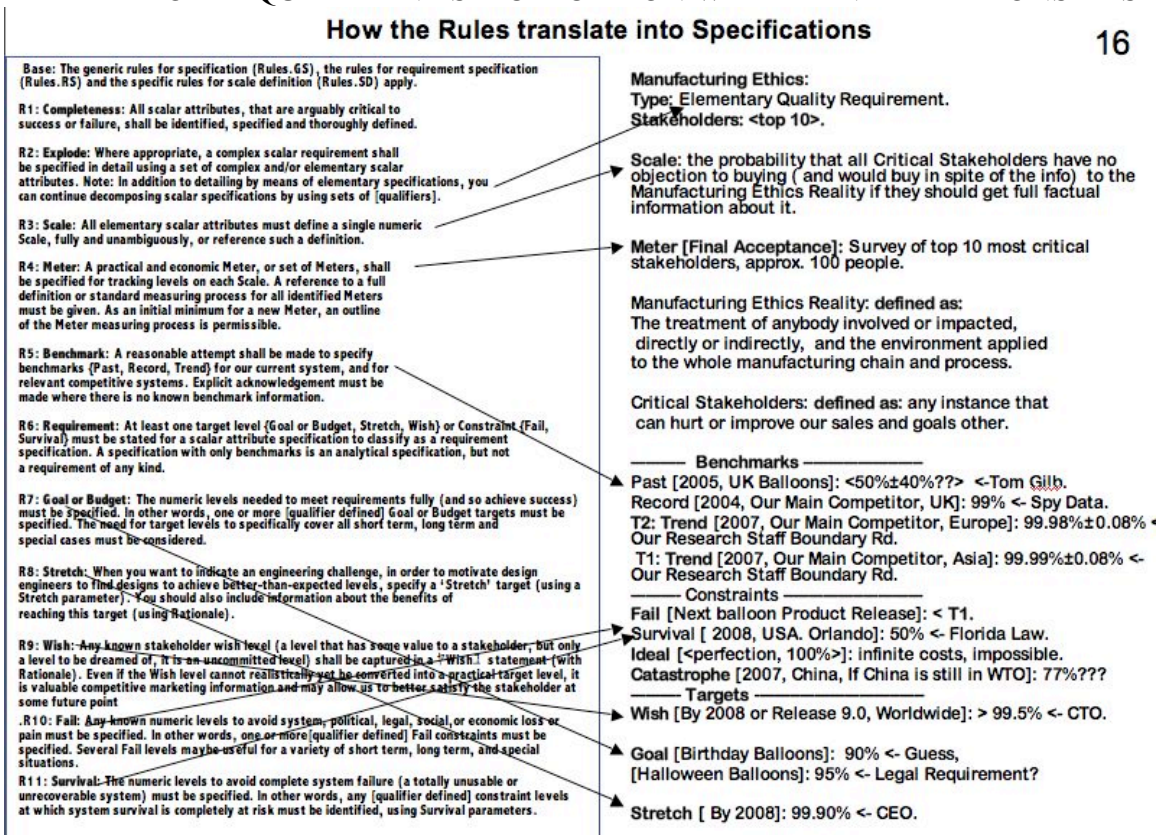
Some Illustrative Classes of Requirement Relationship 'Exploitation Areas' (1, 2, 3, …), and some 'Planguage' specification concept examples (a, b, c, …)

1. Design Support (for the requirement in question)
    a. Impacted By (a design)
    b. Supported By (a design)
2. Requirement Support (for this particular requirement)
    a. Contract Specification
    b. Stakeholder
    c. Stakeholder Need (from which this requirement is derived)
    d. Source
3. Risk Management:
    a. Dependencies
    b. Issues
    c. Risks
    d. Risk Resolution
4. Prioritization
    a. Source
    b. Authority
    c. Funding
    d. Dependencies
    e. Risks
5. Economics:
    a. Estimated Development Cost.
    b. Cost Uncertainty.
    c. Evidence (for cost estimate).
    d. Source of Evidence
    e. Credibility Level
    f. Impact (% of Target, Real level)
6. Development Support: they help us in projects, and in maintenance of systems.
    a. Contract
    b. Test Cases

   c. Tester
   d. Implementation Owner
   e. Specification Owner
   f. Release
7. Systems Analysis
   a. Benchmarks: Past, Record, Trend, Wish
   b. Scale (of measure definition)
8. Project Control
   a. Constraints: Survival Level, Fail Level
   b. Targets: Goal level, Stretch level
   c. Qualifiers (for Levels): (when, where, event)
   d. Meter (specified test processes in development and handover)
9. Architecture Control
   a. Constraints: Survival Level, Fail Level
   b. Targets: Goal level, Stretch level
   c. Qualifiers (for Levels): (when, where, event)
   d. Estimated Impacts (Impact, ± Uncertainty).
   e. Design Constraint

The classes (1, 2, … 9) of requirements relationship information above are themselves not finite. Any useful set of categories could be used. The above list illustrates some possibilities. Similarly, the examples of Planguage specifications (a, b, c, …) are not exhaustive, for any one relationship class, but are illustrative of the specification tools we *can* apply. The Competitive Engineering handbook [CE] is voluminous in defining and exemplifying these concepts in detail.

EXAMPLE OF REQUIREMENT SPECIFICATION WITH MANY RELATIONSHIPS



Example: on the left hand side we show a realistic set of rules (from [CE]) that are a best practice standard for scalar requirement specification, and can also be used for quality control of a specification. One the right hand side we see a teaching example of using the rules to define a quality requirement, together with a number of associated relationships. The 'actual requirement' might be thought of as the 'Goal' statement, near the bottom. All other specification elements could be classified as requirement relationship parameters in Planguage. This is nowhere near a complete set of possible relationships for a requirement (see more that could be added here in the template below). But it is close to common industrial practice with our clients. [Source CE Teaching slides, 2006].

The Qualifier Conditions as Relationships

Notice, in the example above, a statement like:

"Catastrophe [2007, China, If China is still in WTO] 77%"

The '*[2007, China, If China is still in WTO]*' *is called a 'qualifier' in Planguage.*

A statement 'qualifier' has a *list of conditions* that all must be 'true', for the statement to be 'effective'. 'Effective' means to *be* a valid and effective requirement at a given moment, for example, as opposed to a requirement *specification* that is NOT presently valid in all qualifier conditions. Not effectively 'due' at a given evaluation moment. In this example, the year *must* be '2007' (a 'deadline' concept), the Country *must* be 'China', and a *condition* is that China is still in the World Trade Organization. Clearly, all qualifier conditions describe important *relationships* for this constraint requirement.

'Catastrophe' (in the example above) is a 'really bad' level to be at, as the name implies: essentially total and irrecoverable failure of the system.

Implied Relationships in A Scale Definition
Notice in the above example,

Scale: the probability that all Critical Stakeholders have no objection to buying (and would buy in spite of the info) in to the Manufacturing Ethics Reality if they should get full factual information about it.

Notice that there are some clearly specified relationships here with:
• Critical Stakeholders
• Manufacturing Ethics Reality
• factual information

This Scale-of-measure definition is automatically reused for a large number of specifications below it, that need a consistent scale of measure definition, and don't want to repeat the detail of it. For example, the parameters *Goal, Past, Fail, Trend, Wish, Stretch, Meter*.

So, this means that all those requirement specification parameters have a predefined *relationship* to their Scale.

We can go one step further. And I almost invariably practice this in daily industrial use. We can parameterize the Scale itself:

Scale: the probability that all defined [**Stakeholders**] have no objection to buying (and would buy in spite of the info) in to a defined [**Ethics Statement**: default = the Manufacturing Ethics Reality] if they should get defined [**Info**] about it.

This statement makes it clear that this requirement has a relationship to three major *classes of* things:
• **Stakeholders**
• **Ethics Statement**
• **Info**

These classes are generic ideas, and can be defined sets, like:
Stakeholders: defined as: {Buyers, Product Reviewers, Official Government Product Approvers, Corporate Purchasers, Individual Purchasers, Standards Approvers }.

And the specific relationship can be defined in any other statement, using a suitable qualifier:
Goal [Stakeholders = Buyers, Ethics Statement = All Our Corporate Policies, Info = Full Non-Confidential Web Disclosure, Deadline = End 2008, Market = Asia, IF = WTO member] 80%

The simplicity and flexibility of this structure makes it possible to define very complex requirement relationships in an extendible (new relationships later), direct, automatically traceable ("display all 'Stakeholders = Buyers' requirements") and intelligible, way. Here below is a template ('with hints') of the type we use to remind engineers about useful, corporate, encouraged, specification process options, for describing some relationship for a function requirement:

TEMPLATE FOR FUNCTION SPECIFICATION <with hints>

Tag: <Tag name for the function>.
Type: <{Function Specification,
Function (Target) Requirement,
Function Constraint}>:

Note: By default, a 'Function Requirement' is assumed to be a 'Function Target'.

========== **Basic Information** ==========

Version: <Date or other version number>.
Status: <{Draft, SQC Exited, Approved}>.
Quality Level: <Maximum remaining major defects/page, sample size, date>.

Stakeholders: <Name any stakeholders with an interest in this specification>.
Owner: <Name the role/email/person responsible for changes and updates to 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? >.
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>.

========== **Specific Budgets** ==========

Financial Budget: <Refer to the allocated money for planning and implementation (which includes test) of this function>.

Example: this electronic template [CE, Function Chapter] will give some examples of requirement relationships for a non-scalar requirement like a function. The text in <brackets> is a deletable 'hint' as to how to fill out the required information, if you choose to use the parameter. This is not a complete list of

Planguage parameters that could be used to describe relationships. In addition to textbook parameters defining Planguage, the process user is at liberty to invent, on the spot, any useful relationship parameters, independently of any initial Planguage textbook suggestions. For example, we could add something like 'Outsourced To:' to the list above if relevant.

Relationship of Requirements to their Supporting 'Design' using Impact Estimation Tables:

Design Ideas **Requirements**	Central	Youth	Facts	London	Diploma	Events	Discounts	**Sum for Requirement**
Performance Requirements								
Participation	80% ±50%	60% ±70%	0% ±50%	0% ±50%	30% ±50%	20% ±50%	30% ±50%	220% ±370%
Representation	80% ±50%	80% ±50%	10% ±50%	0% ±50%	10% ±50%	20% ±50%	50% ±40%	250% ±340%
Information	0% ±50%	20% ±40%	80% ±50%	0% ±20%	20% ±50%	0% ±50%	0% ±30%	120% ±290%
Conviction	0% ±10%	20% ±50%	60% ±30%	80% ±50%	10% ±50%	80% ±50%	0% ±50%	250% ±290%
Influence	0% ±50%	40% ±40%	60% ±50%	0% ±50%	80% ±50%	80 %±5%	0% ±50%	260% ±340%
Fun	50% ±50%	40% ±50%	10% ±50%	0% ±0%	0% ±0%	80% ±50%	0% ±0%	180% ±200%
Sum of Performance	210% ± 260%	260% ± 300%	220% ± 280%	80% ± 220%	150% ± 250%	280% ± 300%	80% ± 220%	
Resource Requirements								
Financial Cost	20% ±30%	1% ±1%	1% ±1%	1% ±1%	1% ±5%	30% ±50%	30% ±50%	111% ±135%
Performance to Cost Ratio	210/20	260/1	220/1	80/1	150/1	280/30	80/30	

Figure: An Impact Estimation Table (teaching example, here) can be used to chart a many-to-many relationship between any set of requirements, and any set of corresponding designs. This can be repeated at any levels of system consideration, from top to bottom of a system. It can be used to relate systems within a 'systems of systems'. The basis for the estimates should be specified in annotations to each estimate. Source: CE. Details of the impact estimation method will be found in CE and in papers at Gilb.com. In simple terms a 100% impact means that the means is estimated to reach the required Goal level 'on time' (more specifically, with respect to all qualifier conditions including deadline.).

Summary:
The complexity of systems engineering would seem to require a far more precise and detailed specification of the *relationships* a requirement is known to have to all other system concepts. The planning language , 'Planguage', has been developed to cope with this need in a simple way.

The advantage of this extension to conventional requirement specification methods is that we encourage the engineer and systems architect to gather data on these relationships early, rather than later downstream. This should reduce costs and delays caused by much later recognition of the relationships.

Planguage, used as a relationship language, effectively encourages persistent 'specification existence' of systems analysis information, so that it is not lost; for fruitful use in the requirements.

It is 'always' available when conditions change, to help us make smarter decisions about design, architecture, contracting, risks, priorities, and project management.

It serves as a better and more formal system-wide memory of critical relationships.

Systematic rules and conventions, such as those illustrated from Planguage, will enable more-automatic use of requirements, even with simple text string searching.

The language used does not require any permissions or special tools, and will work both with simple text processors and more-advanced requirements tools.

Planguage will allow and support generation of any text view or graphical view we might find useful, from the raw requirement information.

References:

CE: Gilb, Tom, Competitive Engineering, A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage, ISBN 0750665076, 2005, Publisher: Elsevier Butterworth-Heinemann. A sample chapter will be found at Gilb.com.

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

NASA 97:William M. Wilson, **EffectiveRequirements Specifications**
http://satc.gsfc.nasa.gov/support/STC_APR97/write/writert.html

INCOSE 01: **Can Requirements Specifications Be Replaced By Databases?**
http://www.incose.org/symp2001/archive/program/panels/p3_6.html

Raytheon06: *Rick Steiner:* Systems Modeling Language (SysML) and Mission Assurance
How Raytheon's role in defining SysML can lead to better systems
http://wwwxt.raytheon.com/technology_today/2006_i1/feature_3.html

Author Bio:

Tom Gilb is an international consultant, teacher and author. His 9th book is '**Competitive Engineering**: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage' (January 2005 Publication, Elsevier) which is a definition of the planning language 'Planguage'.

He works with major multinationals such as Bosch, Qualcomm, HP, IBM, Nokia, Ericsson, Motorola, US DOD, UK MOD, Symbian, Philips, BAe, Intel, Citigroup and many others. See www.Gilb.com for much more detail, and free publications on Planguage.

His **Planguage** is a major innovation in systems engineering planning. It has been **adopted** by several major multinationals. Philips Medical Systems uses it widely. Intel has trained over 6,000 engineers in Planguage. Citigroup has adopted it for IT projects. HP made an early version of it the corporate standard for product quality specification. Symbian has adopted it for marketing and technical requirements.
Contact: Tom @ Gilb . c o m