Planguage Concept Glossary

Glossary Introduction

Purpose of the Glossary

This glossary contains the master definitions of the fundamental Planguage concepts. Its central purpose is to define 'concepts' – not words. I view this concept glossary as a central contribution of this book, standing in its own right.

"What's in a name? That which we call a rose, by any other name would smell as sweet."

Shakespeare, Romeo and Juliet, Act 2

"Every word or concept, clear as it may seem to be, has only a limited range of applicability."

Werner Heisenberg[1]

With the Heisenberg quotation in mind, this glossary will try to give the interpretation Planguage intends, when the glossary terms are used in *this book*. (If the text and the glossary do not seem to agree, I suggest you trust the glossary primarily as a correct interpretation.[2])

Further explanation of the glossary-defined concepts is found in the main text (via the index). An updated and extended Planguage Glossary is also to be found on the website www.Gilb.com and at www.books.elsevier.com. Space limitations within the book meant that not all the glossary could be included.

Development of this Glossary

I have not tried to define *all* possible concepts for a systems engineering discipline. I have merely concentrated on defining those that I have found useful in my work.

Some other concepts have been included because the glossary has been developed in connection with drafting future books in this Planguage

[1] Heisenberg, Werner, 1958, *Physics and Philosophy*, London: Penguin Books (2000), ISBN 0-141-18215-6, 176 pages.

[2] I believe Bertrand Russell (1872–1970) said that if the experts disagree, you cannot be sure that either one of them is right. So, my advice to trust the glossary must be taken with caution!

series. (The intended titles are *Requirements Engineering, Priority Management* and *Evolutionary Project Management*. Unpublished versions and drafts of these are to be found on the website www.Gilb.com.) So, the glossary may seem somewhat detailed in the context of a single book. But, the intention is to have a common glossary across all the books in the series.

Development of Concepts

In defining a concept, I have not attempted to blindly follow any single particular standard, such as INCOSE, ISO or IEEE. Indeed, I regularly found them inadequate for the specialized purpose at hand. I have primarily tried to let the concepts suit my narrow 'systems engineering' purposes and, above all, to be consistent with each other.

It is worth explaining that I have had considerable help and feedback from my editor and a number of colleagues, correspondents, friends, students and clients regarding definitions, and the choice of primary terms. I have served as a final subjective decision-maker because in language there is no right or wrong, but it is central that the reader know what the writer intends.

I do view the glossary as open-ended for both my own purposes and for purposes of the reader. I also view each concept as potentially capable of continuous improvement in definition.

The Glossary as a Reader-Extendible Tool

I do not mean to impose my terms or definitions on the reader. I respect their rights and needs to define things, in any useful or traditional way for them. I also respect their right to rename any terms. I just needed to take a position on concepts and terms in order to communicate and develop my own ideas. I intend to develop the glossary as needed, and the reader should feel free to do the same, for their own uses and benefits.

The deeper I have gone into this glossary, the more humbled I have been with the infinite possibility of improvement. So, I beg the readers to accept the many imperfections as the best I could do within the timescales, and still publish it in book form at all. I promise to continue the improvement, to participate in improvements and to make this basis freely available at no cost or restriction to people who want to improve it or make it specialized for their own purposes. Permission is hereby given to quote from the glossary freely, and partially, provided suitable credit is given as to origin (© credit to Tom Gilb is sufficient). Notification of your use would always be interesting to me, and may result in useful updates and feedback to you. (Notification of use and reference to Tom@Gilb.com).

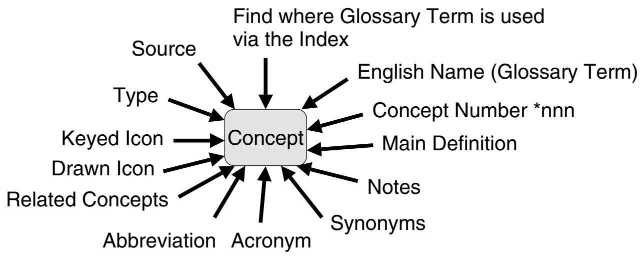
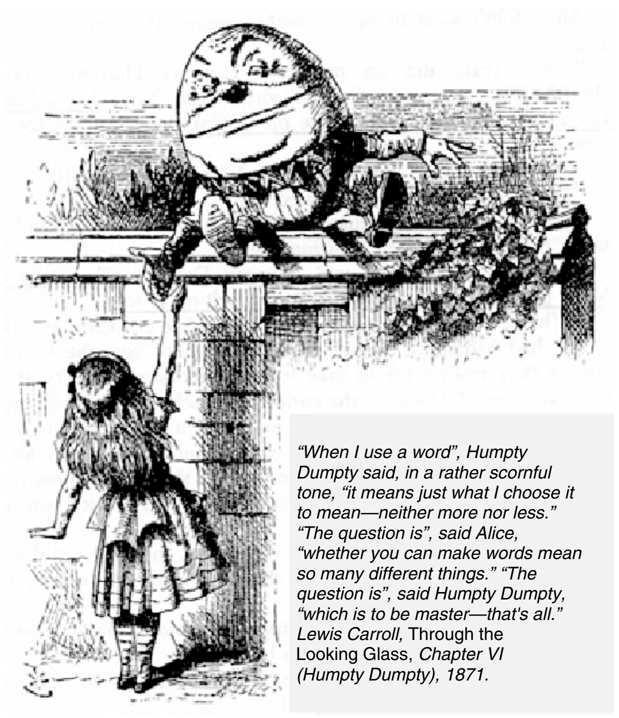


Figure G1

About the Glossary Concepts

A concept can have many 'names' (or 'tags' in Planguage), which act as pointers to it: the names do not change or determine a concept, but merely *cross-reference* it. The central, universal identification tag of a concept is its unique concept number, prefaced by an asterisk, *nnn (for example, *001). The idea behind the concept numbers is to allow and enable full or partial translation into various international languages and into corporate dialects.



"When I use a word", Humpty Dumpty said, in a rather scornful tone, "it means just what I choose it to mean—neither more nor less." "The question is", said Alice, "whether you can make words mean so many different things." "The question is", said Humpty Dumpty, "which is to be master—that's all." Lewis Carroll, Through the Looking Glass, *Chapter VI (Humpty Dumpty), 1871.*

Figure G2
Alice meets Humpty Dumpty[3].

[3] Illustration by John Tenniel to Chapter 6 of 'Through the Looking-Glass' by Lewis Carroll. Wood-engraving by Thomas Dalziel. Illustration from http://www.scholars.nus.edu.sg/landow/Victorian/graphics/tenniel/lookingglass/6.1html/. Additional Lewis Carroll text was added in.

After **Concept *313**

'After' is used to indicate a planned sequencing of events, including Evo steps and tasks.

Example:
Product Trials: Step: {F2, F1 After F2}.
This means that Product Trials (a tag) is a defined Evo Step consisting of step elements F2, and then F1.

Aim **Concept *001**

An 'aim' is a stated desire to achieve something by certain stakeholders. An aim is usually specified informally and non-numerically.

Example:
Our aim is to be the dominant supplier of mobile phones in China by the end of the decade.
Note the two constraining qualifiers (China, By End of the Decade) and the function area (Supplying Mobile Phones).

"The aim must include plans for the future."
<-(*Deming 1993 Page 51*)

"It is important that an aim never be defined in terms of activity or methods. It must always relate to how life is better for everyone."
<-(*Deming 1993 Page 52*).

"The aim precedes the organizational system and those that work in it. Workers, for example, can not be the source of the aim, for how would one know what kind of workers to choose?"
<-(*Deming 1993 Page 52*)
Attributed to Deming by Carolyn Bailey.

Notes:
1. When using the term 'aim,' the intent may be to simplify, and give the ambition level.
2. An aim is ultimately specified in the complete and detailed requirements specification.

   Example:
   "Our aim is to have the <best> book on <gardening>."
   Aim [New System X]: Superior long-term competitive edge in all market areas and product lines.

Related Concepts: Goal *109; Budget *480; Target *048; Ambition *423; Mission *097; Vision *422.

Ambition **Concept *423**

'Ambition' is a parameter, which can be used to summarize the ambition *level* of a performance or resource target requirement. Ambition must state the requirement concerned (like 'Usability') and it must contain a notion of the kind of level being sought (like 'high').

Notes:
1. The Ambition summary is useful for getting team understanding and agreement to its concept, before going on to the detailed specification work. It can then be used during development of the specification as a basis for judging the relevance of the details. The

Ambition can also be updated to reflect the detailed specification better, if desired.

2. Once the specification is completed, Ambition provides a useful overview summary of the more detailed specification.

Example:
Usability:
Ambition: The system will be extremely/competitively easy to learn, and to use, for a variety of users and user cultures.
Reference: Quality Attribute Usability Paper, Version 0.2.

Keyed Icon: @.Σ "Target and Summary."

And — Concept *045

'And' is used as a logical operator to join any two expressions within a statement.

Example:
Goal [If War and Inflation]: 60%.
Goal [If Peace And Inflation]: 60%.
Goal [If War AND Stability]: 60%.
To make a statement read better, the lead capital letter can be dropped, giving 'and' rather than 'And'.

Architecture — Concept *192

The 'architecture' is the set of components that in fact exist and impact a set of system attributes directly, or indirectly, by constraining, or influencing, related engineering decisions.

Note: Interesting specializations:

- **Perceivable Architecture**: the architecture, which is somehow directly or indirectly perceivable in a real system, as determining the range of performance and cost attributes possible. This applies regardless of whom, if anyone, consciously specified the architecture design artifacts.
- **Inherited Architecture**: the architecture, which was not consciously selected *for this system* at a particular level of architecture activity, but was either:
  – incidentally inherited from older systems,
  – accidentally inherited from specified design artifacts, specified by architects, managers or engineers.
- **Specified Architecture**: the formally defined architecture specifications at a given level and lifecycle point, including stakeholder requirements interpretation, architecture specification, engineering specification done by this architecture level, certification criteria, cost estimates, models, prototypes, and any other artifact produced as a necessary consequence of fulfilling the architecting responsibility.

Note: an extensive discussion of the architecture concept is given in Maier (2002), including a special appendix on the history of attempts to define a standard in DoD, IEEE, INCOSE, Appendix C pp. 283–289. In addition the book gives a great many other insights into the nature of the concept.

Note: The highest *specified* level of design ideas for a defined system is called the 'architecture'. The architecture is the collection of controlling design ideas for a defined purpose. The architecture refers primarily to frameworks, interfaces and other technology and organizational ideas which more-detailed design ideas are expected to fit in to.

Notes:

1. The architecture specifications (*617) would probably be classified as generic design constraints (or 'architecture constraints', if you wanted to emphasize the idea of 'architecture').
2. *Architecture* specifications would have priority over subsequent design decisions, made at more-specialized engineering levels.
3. 'Architecture specification' is the set of system-wide decisions, which are made in order to improve the systems survival ability, as it is threatened by changes to it, and by its environment.

"*Architecture*: A high level design that provides decisions about:

- purpose (What problem(s) that the product(s) will solve)
- function description(s) (Why has it been decomposed into these components?)
- relationships between components (How do components relate in space and time?)
- dynamic interplay description (How is control passed between and among components?)
- flows (How does data or in-process product flow in space and time?)
- resources (What resources are consumed where, in the process or system?)"

Source: Standard: FAA-iCMM Appraisal Method Version 1.0 A-19, INCOSE Conference CD, June 1999, Brighton UK [FAA98]

This definition differs from Planguage in that we are primarily concerned with design aspects, and this contains three requirement notions.

"Architecture: The organizational structure of a system or component."

Source: [IEEE 90] in [SEI-95-MM-003] Standard: An example of an IEEE definition of 'architecture'.

Domain: systems
engineering.specification.design.architecture

Related Concepts [Architecture, *192 collective noun]:

- Design (noun) *047
- Design specification (*586, or *047 + *137)
- Design Ideas *047
- Architecture (the process) *499
- Architecture Specification *617
- Artifact *645
- Systems Architecture *564

Keyed Icon *192 Architecture:

- (delta, symbol pyramid architecture).

Note keyed and drawn icon for design (a subset of architecture is a rectangle

- (or [*Design X*]) which is analogous to the blocks used to make the pyramid).

Type: engineering specification type.

Architectural Description [IEEE] Concept *618

Architectural description is ''a collection of products to *document* an architecture.'' (This definition is identical with IEEE Draft Standard 1471, December 1999.)

This concept is generic and can apply to any specific architecture type.

Notes:

1. The intentionally broad term 'products' is used to include *anything*, which might be useful in describing an architecture. *Anything* can

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

include physical models, computerized models, prototypes, blueprints, parts lists, planned test results, actual input and outputs from tests, Planguage architecture specifications, sales and training materials, and real systems – as long as their *purpose* is to document an architecture.

2. The term 'Architecture Description' is an IEEE term, it is NOT used in the Planguage sense of a 'Description' parameter: it should really be equated to the Planguage term, 'Definition.')

Related Concepts:

- Architecture Specification *617: This concept does not include models and real systems, but only abstract specifications
- Systems Architecture *564: An architecture description can be for any specialized subset of a systems architecture, such as software or hydraulics.
- Architecture *192: this is the real set of artifacts that the architectural description describes.

Architecture Engineering — Concept *499

The architecture engineering process puts in place the systems architecture, which is a controlling mechanism for the design engineering of any project.

Architecture engineering defines the strategic framework (the systems architecture), which design engineering has to work within. It lays down the standards, which help control such matters as the tradeoff processes amongst requirements. It helps synchronize design engineering disciplines across different systems.

The architecture engineering process is a *subset* of the Systems Engineering process.

Notes:

1. The architecture engineering process is distinct from the larger systems engineering process in that it is focused on *design issues*. (Systems engineering is broader. It includes consideration of the requirements, quality control, project management, and any other discipline, that is useful for satisfying requirements.)
2. The architecture engineering process is distinct from the other system level design engineering processes because it operates at a higher level, and is therefore concerned with wider issues. It has to consider the overall strategic framework and provide guidance to all the lower-level systems. It considers especially the long-term objectives, and the totality of the requirements for all systems.
3. The architecture engineering process is, ideally, technologically neutral. It should provide guidance on design, using any relevant technology, policy, motivation, organizational idea, contractual agreement, sales practice and other devices. One of the main criteria is that the architecture is cost-effective. Note that technological neutrality is not always achieved! For example, promotion of the use of standard platforms could be included within a systems architecture; and while that is an architectural decision, it is not technologically neutral.

Synonyms: Architectural Engineering *499; Architecting *499.

Related Concepts: Systems Architecture *564; Architecture *192; Requirement Engineering *614; Design Engineering *501; Architecture Specification *617.

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

Architecture Specification **Concept *617**

An architecture specification is the written definition of an architectural component.

Notes:

1. An architecture specification either specifies a component of a systems architecture, or it specifies an architectural component of a specific system.
2. An architecture specification is a specialized form of design specification.
3. Architecture (the collective noun) is the *real* set of artifacts that the architecture specification describes. In other words, this is the *observable* architecture in a defined system. The specification may be describing *desired future* states of that system. Some parts of that specification might *never be implemented* in practice, since it serves as a vehicle to discuss architectural possibilities and options.
4. An Architecture Specification is not as broad as an Architecture Description [IEEE], which can also include models, prototypes and real systems to aid architectural description.

Synonyms: Architectural Specification *617.

Related Terms Architecture *192; Architecture Engineering *499; Systems Architecture *564; Architecture Description *618.

Assumption **Concept *002**

Assumptions are unproven conditions, which if not true at some defined point in time, would threaten something, such as the validity of a specification or the achievement of our requirements.

'Assumption' is a parameter that can be used to explicitly specify any assumptions made in connection with a specific statement.

> "Assumptions are suppositions, conjectures, and beliefs which lack verification at the time of writing, or requirements and expectations that are not within our power to control, but which have been used as part of the basis for planning future actions. We identify for each the degree of risk involved and possible consequences if the assumption is erroneous."
>
> *<-Don Mills, NZ 2002 (Personal e-mail)*

Notes:

1. We need to document our assumptions systematically in order to give warning signals about any conditions that need to be evaluated, or checked, to ensure that a specification is valid. The aim is that the assumptions will be considered at the relevant future points in time, and that anyone with any additional information concerning an assumption (including lack of specification of an assumption), will volunteer it as soon as possible.
2. The purpose of the Assumption parameter is to explicitly state 'otherwise hidden' or undocumented assumptions. This permits systematic risk analysis.
3. There are many different ways in Planguage to express assumptions. Alternatives to using the Assumption parameter include using the Rationale, Condition and Basis parameters.

Example:
Hierarchic Structure [Health and Safety System]:
Type: Design Specification.
Description: A hierarchical database structure will be used.

Assumption: No negative impact on performance of Emergency and Rescue Inquiries <- JB.
Impacts: Access Response, Portability.
Is Impacted By: Available Database Packages.
Rationale: This structure is compatible with the current structure, and can be directly converted to it.
Condition: Off-the-shelf software can be used, and no in-house support is needed.
Basis: Health and Safety System required by National Law.

4. It would be good practice to specify the *consequences* of a failure for the assumption to be true. Use Impacts, Supports and similar parameters just below the Assumption statement. *See above example.*
5. It would also be good practice to specify the things that *determine* if this assumption is going to be true. Use Depends On, Authority, Source, Is Impacted By and similar parameters just below the Assumption statement. *See above example.*

Related Concepts: Basis *006; Rationale *259; Condition *024; Qualifier *124; Risk *309.

Attribute — Concept *003

An attribute is an observable characteristic of a system. Any specific system can be described by a set of past, present and desired attributes. There are four main categories of attribute:

- Performance: 'How Good the System Is'
- Function: 'What the System Does'
- Resource: 'What the System Costs'
- Design (or Architecture): 'The Means for delivering the System'

All attributes are qualified by Conditions, which describe the time, place and events under which the attributes exist.

Attribute: "A characteristic of an item; for example, the item's color, size, or type."

Source: Dictionary of Computing Terms*, IEEE 630-90.*

Notes:

1. Performance and resource attributes are scalar (described by a scale of measure). Function and design attributes are binary (either present or absent).
2. Attributes can be complex. They can be defined by a sub-set of elementary attributes.
3. An attribute may be described by any useful set of Planguage parameters.

Example:
Reliability: "The attribute tag name."
Ambition: High duration of operation. "*Summary of the target.*"
Scale: Hours of <uninterrupted service>. "*Defining the measure.*"
Goal [Next Release]: 6,000 hours. "The required target level for the attribute."
The tag (Reliability) and the parameters (Ambition, Scale and Goal) provide a systematic framework for defining and referring to a scalar attribute's components.

Synonyms: Characteristic *003; Property *003.

Related Concepts: Performance *434; Function *069; Resource *199; Design *047.

Author **Concept *004**

An author is the person, who writes or updates a document or specification of any kind.

Notes:

1. This is a generic term, which depending on the specific document type, is usually replaced by specific roles, such as {engineer, architect, manager, technician, analyst, designer, coder, test planner, specification writer}.

Synonyms: Writer *004; Specification Writer *004.

Related Concepts: Owner *102.

Authority **Concept *005**

Authority is a specific level of power to 'decide' or 'influence' or 'enforce' a specific matter requiring some degree of judgment or evaluation. For example, the status of a specification is usually the responsibility of some 'authority' (some set of individuals holding the specific authority). Authority is often held by a specified individual or by an organizational group. A specific role may hold the authority. In addition, a document that is authorized can be used, within the document's scope, as a source of authoritative information (in lieu of access to the people holding the authority).

An Authority *parameter* is used to indicate the specific level of authority, approval, commitment, sanction, or support for a specified idea, specification or statement.

Notes:

1. This is not the same as 'Source,' which is the written or oral source of information. A Source might convey no authority whatsoever (for example, ''60% <- My best guess!'').

   Example:
   Past [Last Year]: 60% <- Marketing Report [February, This Year].
   Authority: Marketing Director [Tim].

Background **Concept *507**

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.

Example:
In a requirement specification, the benchmarks (Past, Trend, Record) are not the actual requirements (not 'core'), but they are useful 'background' to the requirements.
The key requirement targets (Goal and Budget levels) and constraints (Fail and Survival levels) are central (core) to implementation, and are therefore not background.

Notes:

1. Parameters are clearly typed as either 'core' (for example, Scale, Meter and Goal) or 'background' (for example, Ambition, Gist and Past), or 'commentary' (for example, Source and Note).
2. Background specifications are essential to the understanding and use of a specification. However, any defects in background will *not necessarily* materially and/or negatively impact the real system. Such defects might potentially have bad impacts when used in a certain way. For example, when a Goal (core specification) is set on the basis of an incorrect Past

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

or Record (background specification) the resulting Goal level (a 'core' specification) will be incorrect.

Specification defects in 'background specification' are either major or minor, depending on our judgment, in the specific context, of the *potential* consequences.

Example: [Background Parameters]

• Benchmarks {Past, Record, Trend} • Owner • Version • Stakeholders • Gist • Ambition

Related Concepts: Non-Commentary *294; Core Specification *633; Commentary *632; Specification *137.

Backroom — **Concept *342**

Backroom is an adjective or noun, referring to a conceptual place, used to describe any processes or activities in Evo that are not necessarily visible to the Evo step recipients.

Notes:

1. Typically, Backroom is used to refer to the development and production cycles of the Evo result cycle.
2. This is where concurrent engineering takes place. Backroom activities (for example, detailed design, purchasing, construction and testing) may have to be carried out in parallel with other activities as step preparation (prior to being ready for delivery), can take arbitrary lengths of time. The overriding Evo requirement is for frequent stakeholder *delivery* cycles.
3. Evolutionary project management needs to manage the backroom and frontroom as one synchronized process.

Related Concepts: Frontroom *343.

Baseline — **Concept *351**

A system baseline is any set of system attribute specifications that defines the state of a given system.

Frequently, the choice of system baseline is governed by project timescales; a significant project milestone date in the past will be selected and then it is simply a case of determining the relevant individual attribute baselines on that date.

An attribute baseline is a benchmark that has been chosen for use as a start point to measure any relative system change (estimated or actual) against.

Notes:

1. Within Impact Estimation, for each scalar attribute a 'Baseline to Target Pair' is declared. The chosen baseline is usually a Past level and represents zero percentage impact (0%). In an IE table, each baseline to target pair appears immediately under the tag of its attribute on the left hand side on the table.

Example:

QX: Quality.

Scale: Time to complete a defined [Task] for a defined [Person Type].

Baseline: Past [Task = Learn to Drive Off, Person Type = Experienced Driver, Our Competitor's Product]: 1 minute.

Target: Goal [Task = Learn to Drive Off, Person Type = Experienced Driver, Our New Product]: 10 seconds.
This example shows setting a baseline and a target for a quality, QX.

Example:
ABC: IE Table [Baseline Date = Nov 7, Target Date = Dec 7].
QX:
BABC: Baseline [ABC]: Past [. . . . . . . "declare as a baseline for ABC IE table"].
TABC: Target [ABC]: Goal [. . . . . . "declare as a target for ABC IE table"].
Baseline to Target Pair [ABC]: 1 minute <-> 10 seconds. "deduced from baseline and target declarations above. Strictly not needed as repetition."
This example shows an alternative way to set a baseline and target. It introduces the idea of declaring a Baseline Date and Target Date applying across an IE table.

Table G1 ABC: IE table.

Design Idea ->	*ADI*	*BDI*	*CDI*	*DDI*
QX				
1 minute<–> 10 seconds	0%	100%	−20%	150%

Design 'ADI' has zero percentage impact, meaning that if Design 'ADI' were implemented then there would be no visible change in the quality level (it would remain at one minute and there would be no forward progress towards the target (10 seconds)).
Design 'CDI' would be even worse than the baseline and the quality level would be worse than before.
Related Concepts: Benchmark *007.

Basis — Concept *006

A basis is an underlying idea that is a foundation for a specification.
A 'Basis' parameter is used to explicitly specify a foundation idea, so that it can be understood and checked. Hopefully, if necessary, a basis specification will be challenged and corrected. It is a tool for risk analysis.
Notes:
1. Basis statements are used to declare a set of conditions, which we assume will be true. We want to make it quite clear that the related statements are *entirely contingent* upon the conditions being true.
   A Basis statement is, or will be, for the appropriate qualifier time, place and other conditions, fundamental and stable. We state a Basis in case it is untrue, or is a misunderstanding, or needs improvement in specification: the intended readership needs to check whether they agree that a Basis statement applies. In addition, we state a Basis to ensure that the conditions are checked later, at the relevant time.

2. Basis is different from Assumption. An Assumption is a set of statements, which we *expect be true* in the planning horizon (for example, the dates indicated in Goal and Fail parameters), but we *cannot be sure*; they can well change. The related specification *may* need updating if they do.
3. Basis is quite different from Rationale. A Rationale is a set of statements, which lead to a desire to make a specific specification. It explains how we got to that particular *specification*. Basis is a set of statements, which are the *foundation* on which a specification is made. If the result of evaluation of any of the relevant Basis statements changes, then the specification may no longer be valid.

Example:
Fail [A1]: 60%, [Not A1]: 50%? <- Guess as to consequence.
A1: Assumption: Drugs Law [Last Year] is still in force and unchanged with respect to this plan.
Basis: Drugs Law 'Conditions for Approval for Human Trials'<- Pharmaceutical Law [Last Year].
Rationale: Our Corporate Policy about following laws, strictly and honestly <- Corporate Ethics Policy.
Condition: Applies only to <adult, voluntary, healthy, field-trial people>.
'A1' is a defined assumption that can be reused in this or other contexts.

Synonyms: Base *006; Foundation *006.

Before **Concept *312**

'Before' is a parameter used to indicate planned sequencing of events, including Evo steps and tasks.

Example:
Stage Liftoff: Step: {Ignition On Before Check Thrust OK, Ignite Motors} Before Release Tie Down.
The Evo step is planned as a sequence of step elements. Ignition On is to be done first. Followed by Check Thrust OK. Ignite Motors can be done anytime in relation to the first two, but, since it is in the brackets, it, as well as the other two events in the brackets, must be done before Release Tie Down.

Benchmark **Concept *007**

A benchmark is a specified reference point, or baseline. There are two main types: scalar and binary benchmarks.

Notes:
1. A scalar benchmark is a reference level for a performance or resource attribute. It is usually used for comparison purposes in requirement specification, design and implementation.
2. A scalar benchmark is normally defined using the benchmark parameters {Past, Record, Trend}.

Example:
Usability:
Ambition: Order of magnitude better than future competitors.
Scale: Average time needed to learn to do Typical Tasks for Typical User.

Trend [Best Competitors, During New Product Lifetime, Europe Market & USA Market]: 5 minutes.
Fail [New Product, All Markets]: 2 minutes.
Goal [New Product, Initial Release]: 1 minute.
Goal [New Product, 1 Year After Initial Release]: 30 seconds.
'Trend' is the benchmark specification.

3. Function and design attributes are specified as binary benchmarks: binary attributes are either present or absent in a system.

Related Concepts: Baseline *351; Past *106; Record *127; Trend *155.

Benefit **Concept *009**

Benefit is value delivered to stakeholders.

Notes:

1. Benefits are the positive things that the stakeholders experience from a system. 'Bene' means 'good'.
2. Benefit differs from stakeholder value. Value is perceived future benefit. Value is reflected in what priority, and consequent resources, people are willing to give for something, in order to get the benefits they expect.
3. Benefit is the reality experienced in practice by defined stakeholders.
4. Benefits can include improved stakeholder environment performance, reduced costs, and improved functionality.
5. Benefits could also include the relaxation of previous constraints.
6. Systems engineering control can only be exercised over benefits, which have been specified as requirements. Reaching and keeping an unspecified benefit is unlikely!
7. Systems engineering can add value, it is up to the stakeholders to actually turn that value into benefit by exploiting the system.
8. One way to measure improvements in benefit is to extrapolate from changes in performance levels.

Synonyms: Gain *009; Profit: Informal use; Advantage: Informal use.

Related Concepts: Value *269; Performance to Cost Ratio *010; Value to Cost Ratio *635; Effectiveness *053; Stakeholder *233.

Binary **Concept *249**

Binary is an adjective used to describe objects, which are specified as observable in two states. Typically, the two states are 'present' or 'absent', or 'compiled with' or 'not complied with.'

Notes:

1. All the non-scalar attributes are binary (that is, the function and design attributes).

Related Concepts: Scalar *198.

Budget **Concept *480**

A 'budget' is a resource target: an allocation of a limited resource. A Budget *parameter* is used to specify a primary scalar resource target.

The implication of a Budget parameter specification is that there is, or will probably be, a commitment to stay within the Budget level (something which is not true of a Stretch or Wish specification).

Example:
Maintenance Effort:

Scale: Total annual Maintenance Engineering Hours per thousand lines of software code supported.
Budget [First Four Years Average]: 10 hours.
Stretch [First Four Years Average]: 8 hours.
Wish [First Four Years Average]: 2 hours.
Fail [Any Single Operational Year]: 100 hours <- Client payment limit in contract §6.7.
A Budget specification, together with 2 other resource targets and a constraint.

Notes:

1. A budget level is often arrived at through a formal budgeting process: the budget levels usually being set with regard to priorities, and available financial resources. Sometimes a budget level is determined by cost estimation, or it is determined by using competitive bidding and contracting. In some cases, the budget is absolutely fixed in advance, and we have to try to keep within it by making requirement tradeoffs or by using 'design to cost'.
2. At the very least a warning signal should be noted when a budgeted level is exceeded by a design, by an evolutionary step, or when there is a risk or threat that the budget *might* be exceeded. For example, we need to react if a resource threat to the budget level is discovered while evaluating potential alternative designs.
3. A resource target is a budget concept (small 'b' for budget). In Planguage, there are several parameters used to specify resource targets {Budget, Stretch, Wish}. The Budget parameter (capital 'B' for Budget) is used to specify the major type of resource target.

Synonyms: Budget Level *480: See Level *337; Planned Budget *480; Plan [Resource] *480: Historic usage only; Planned Level [Resource] *480: Historic usage only.

Related Concepts: Aim *001; Resource Target *436: Synonym is budget (the concept); Target *048; Stretch *404; Wish *244; Ideal *328; Goal *109.

Keyed Icon: > "A single arrowhead pointing towards the future. The same basic icon as for Goal *109, but always use an input arrow to a function oval to represent a resource attribute. In context: --->--->O
The Budget icon is the '>' on the arrow. If other levels for the resource are shown on the same arrow, the positioning of the tips of the icon symbols reflects the levels relative to each other."

Catastrophe — Concept *602

A catastrophe level of an attribute is where disaster threatens all, or part, of a system. Catastrophe can mean a variety of things such as:

- contractual non-payment performance level
- illegal quality level
- totally unacceptable level for defined stakeholders
- a level which causes the entire system to be useless (that is, worse than the survival limit).

The Catastrophe parameter can be used to specify any such known disaster level. Using the Survival parameter is another option. (These two parameters are the two sides of the same level.)

Notes:

1. The default assumption is that the catastrophe is for *the complete system.* If it is not, then qualifiers must limit its scope.
2. If a design or architecture threatens to result in any attribute being equal or worse than its Catastrophe Level, then you would discard the design, abandon or modify the requirement, or potentially abandon the project.
3. A catastrophe is not a transient failure – that is we do not expect the system to recover without some major intervention.
   Catastrophe does not imply complete irrecoverable failure. After the event, someone might change their mind and decide to 'bail out' the system. But Catastrophe, once reached, is most likely to be irrecoverable in practice.
4. A Catastrophe Range starts from the 'best' Catastrophe Level and goes in the direction of 'worse'. This can be made explicit by describing the Catastrophe Range, not just the Catastrophe Level (Describe the range by using 'or less' or 'and worse' after the numeric value).
   Example:
   Catastrophe [System Wide]: 60% or less.
   Catastrophe [Security]: 60%.

Keyed Icon: . "A full stop. In context, a series of '.' indicates a Catastrophe Range ------->--!--------] . . . .>O . . . . [-----!-->----------->
The Survival icon (square brackets on a scalar arrow icon . . . . [----] . . . . >) can be used to emphasize the transition from Survival status to Catastrophe (non-survival) status."

Synonyms: Catastrophe Level *602; Catastrophe Limit *602; Intolerable *602; Catastrophic Failure: Informal use only; Death: Informal use only; Non-Survival: Informal use only.

Related Concepts: Survival *440; Range *552: For description of 'Catastrophe Range.'

Historical Note: The idea for Catastrophe originated from Terje Fossnes and Cecilia Haskins in August 2002.

Checking Rate — Concept *015

The checking rate is the *average* speed at which a specification is checked by a checker, using all the relevant related specifications and standards {the main specification, rules, checklists, source documents and kin documents}.

Notes:

1. The checking rate is critical for Specification Quality Control, and must normally be about 300 significant words (of checked main specification) per hour. This can vary (0.1 to 1.9 hours per 300 significant words), depending on many factors, such as the number of documents to be referenced while checking. The optimum checking rate is the checking speed that in fact works best for an individual checker to do their assigned tasks.

Related Concepts: Rate *139; Optimum Checking Rate *126.

Checklist — Concept *016

A 'checklist' for SQC usually takes the form of a list of questions. All checklist questions are derived directly and explicitly from cross-referenced specification rules. Checklists are 'stored wisdom' aimed at helping to interpret the rules and explain their application. Checklists are used to increase effectiveness at finding major defects in a specification.

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

Example:
STDQ: Rule: All critical project requirements must always be expressed numerically and measurably.
This is the rule. The associated checklist question is designed to help people understand how to apply the rule in practice, and identify any defects breaking the rule.
Checklist Q: Are all *performance* concepts (including all qualitative concepts – all '-ilities') expressed *quantitatively*? <- Rule.STDQ.
An example of a checklist question with the rule it supports (STDQ) being referenced.

Notes:
1. Checklists are like law court interpretations of the law. They are not the official 'law' itself, but they do help us understand the proper interpretation of the law. Anyone can write checklists at any time to give advice on how to check. They are intentionally less formal to create, and to change, than specification rules. They do not necessarily have formal 'owners.'
2. Checklists should not be used instead of a proper set of rules, which is maintained by an engineering process owner. They are only intended as a supplement for checkers. Issues can only be classified as real defects if they can be shown to violate the official agreed rules for a specification.
3. Less formal 'de facto checklists' also exist. These include any documents that can be used to check a document with a view to identification of defects. These can have other names and even other purposes than a 'pure' checklist. Examples of 'de facto checklists' include 'sources,' 'standards,' 'guidelines,' 'templates' and 'model documents.' If they help check, they must be some sort of checklist, irrespective of what people call them or intended them to be used for.

Commentary — Concept *632

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.

Example:
• Note • Comment • "Text in quotes" • Source

Related Concepts: Non-Commentary *294; Core Specification *633; Background *507; Specification *137.

Complex — Concept *021

A complex component is composed of more than one elementary and/or complex component.

Notes:
1. A complex component consists of several sub-components. The sub-components may be all of the same type as the component, or of several different types.
2. Requirements, Design Ideas and Evo Steps are often complex components.

   Example:
   Goal [Alpha]: 30%, [Beta]: 20%. "This is a complex statement."
   Goal [Theta]: 50%. "This is an elementary statement."

Related Concepts: Elementary *055; Component *022.

Concept [Planguage] **Concept *188**

A Planguage concept is a formally specified idea used in Planguage.

Notes:

1. There are several types of concept found in Planguage specification. These include:
   - formal Planguage concepts defined in this glossary or other Planguage glossaries and assigned a concept number (*nnn). Some concept names are written with a Capital letter first, to signal that they are formally defined terms. Examples: Scale, Goal, and Defined As.
   - user-defined terms.
2. A Planguage concept, once defined, can be referenced by any useful synonyms or identifiers. These include tags, keyed icons, drawn icons, abbreviations, synonyms, acronyms and alternative language terms (for example, German or Japanese terms).
3. The central idea of a Planguage concept is that the concept itself is independent of the particular means (pointer, reference, cross-reference, tag, icon, concept number) that we choose to apply in order to reference that concept. We can focus on the concept, and not the particular term, about which people might disagree or have cultural difficulties in accepting.
4. Defined concepts can be:
   - reused without explaining them again
   - redefined by Planguage users locally (which simultaneously changes (hopefully improves) the definition of all the other terms, which reference the defined concept)
   - referenced by a set of terms in any language, without necessarily having to rewrite the concept definitions themselves in that language. For example, the concepts could be defined in English, but a Norwegian set of pointers to the concepts can be quickly defined, to permit teaching or multinational project learning and use of specifications.

Example:
Begrep [Norwegian Bokmål] = *188 "Concept [Planguage, US]."
Tilstand [Norwegian Bokmål] = *024 "Condition [Planguage, US]."
Marked [Norwegian Bokmål] = Market [Corporate Glossary].
Is [Norwegian Bokmål] = Ice Cream [Project XYZ Glossary].
*In these examples Planguage concepts like *188 are given a foreign language name ('Begrep') in Norwegian. Using the synonym, 'Begrep,' a user can access the full definition in English.*

Synonyms: Planguage Concept *188.
Related Concepts: User-Defined Term *530; Planguage *030.

Condition **Concept *024**

A condition is a specified pre-requisite for making a specification or a system component valid.

Notes:

1. Evaluation of the status of a condition can be carried out anytime, and on many different occasions, each with a potentially different result.

The result of an evaluation of a condition is the 'current condition status' or more simply, 'status.'

2. *Evaluation* of a condition will determine if its status is currently true or false.
3. There are several distinct kinds of conditions:
   - Reusable Conditions
   - Qualifier Conditions
   - Pre-requisite Conditions

Reusable Conditions:

The Planguage *parameter* 'Condition' is used to define *conditional terms*. The 'true or false' status of such a term can be determined when required. This parameter statement can be used to define:

- reusable conditions (conditions that many other statements can make use of)
- conditions which are complex, and get simplified by having a single tag to express them.

Example:

Senior: Condition {Senior Citizen Or Service over 20 years to Company}.
Pass Through: Condition: Traffic Light {Green, Yellow, Blinking Yellow, Not Red}.

Qualifier Conditions:

One or more qualifier conditions can be used to specify a statement *qualifier* (for example, '[End of March, USA, If Peace]'). A statement qualifier must be completely true for the qualified statement to be valid.

Example:

Level X: Goal [A, B, C]: 33%.
Note: Level X is only a valid Goal when all three qualifiers {A & B & C} are true/valid.

Another example, a Goal level specification is only valid when all the conditions in its qualifier are true. The qualifier in the Goal statement below has three conditions.

Example:

Goal [Year = Release + 1 Year, Market = Europe, Not War]: 66%.

A qualifier condition may consist of an explicit tag name with an appropriate variable declared (for example, 'Market = Europe.' 'Market' is the tag name and 'Europe' is the variable).

If there is no ambiguity, the tag name may be implied and simply the variable is stated.

A qualifying condition may, or may not, be satisfying a Scale qualifier.

Example:

Learning Time:
Scale: Time in minutes for a defined [Role] to <learn> a defined [Task].
Goal [Task = Login, Role = Operator, Country = Spain]: 2 minutes.

In the above example, '[Task = Login, Role = Operator]' is a statement qualifier.

Both 'Task' and 'Role' are qualifier conditions. They are also both Scale qualifiers. Task is assigned a variable of 'Login,' and 'Role' a variable of 'Operator'.

'Country = Spain' is an additional qualifier condition, which has been added. It is not a Scale qualifier.

If the Task under consideration is 'Login,' the Role is 'Operator' and the Country is Spain, then the target goal for consideration must be 2 minutes. In other words, the evaluation of the statement qualifier as 'true' depends on all its qualifying conditions being 'true.' Each qualifier condition is only true if its variable matches the specific instance being considered (Task is 'Login,' Role is 'Operator' and Country is 'Spain'). Each qualifier condition might have a set of valid variable settings. For example, Country: {Spain, USA, Germany}.

Pre-requisite Conditions:
A set of conditions, can be used as a prerequisite for a system component, such as entry to a defined process, or exit from a defined process, or use of a product. Any such conditions should be explicitly listed as pre-requisites or qualifications.

Example:
Exit Conditions:
X1: Senior. "See definition in above example."
X2: Level X. "Not only A & B & C, but also 33% Goal reached."

Example:
Process: Evening Closedown [Application: Default: ABC].
"The square brackets, '[ ]', specify a qualifier condition. It asks the question: Which application is this generic process being applied to?"
Gist: Application process for evening closedown for the night.
Entry Conditions:
E1: All users have logged off. "A condition. Are all users logged off: true or false?"
E2: After 8pm. "Another condition. Is time after 8pm: true or false?"
Procedure
... "If all entry conditions are met (that is, are 'true'), then it is 'valid' to carry out the process."

Synonyms: Conditional Term *024; Pre-Requisite Condition *024.
Related Concepts: Condition Constraint *498; Qualifier *124; Status *174: The result of the evaluation of a condition.
Keyed Icon: [<*condition tag name*>]

Condition Constraint — Concept *498

A condition constraint is a requirement that *imposes a conscious restriction* for a specified system scope. A condition constraint, also called a 'restriction,' is a binary type of *requirement*.

Notes:

1. A condition constraint differs from a 'condition' in that some kind of failure, invalidity, problem, dependency, risk, or other problem may be experienced, if the constraint is not met. It serves as a warning signal for problems.

   Example:
   CCR: Constraint [Release 1]: Initial product must be delivered before the end of January.
   Rationale: Financial penalties apply if this contractual deadline is not met <- Contract Section 2.4.

2. A condition constraint can be categorized by innumerable useful categories, but some common ones are design, legal, cultural, market,

geographic, safety, and language. (Note these categories can also apply to other types of constraint, for example, a certain level of reliability – a scalar performance constraint – could also be a 'legal constraint').

Example:
C1: Constraint [Language]: All official languages of a market will be fully supported in the user interface, and all training and handbook information.
C2: Constraint [Safety]: All Electrical Equipment brought onboard any Corporate Aircraft as Standard Kit will comply with Corporate Electrical Safety Standard 1.5.

Synonyms: Restriction *498.
Related Concepts: Condition *024; Constraint *218; Status *174: Synonym is 'State'.

Consists of — Concept *616

'Consists Of' is a parameter used to list a *complete* set of the subcomponents or elements comprising a component.

Example:
Security:
Consists Of: {Integrity, Attack}.
Alternatively, Security = {Integrity, Attack}.

Related Concepts: Includes *391 "Used to list *some*, but not necessarily all components"; Element *022.
Keyed Icon: = {...} "Is equal to the set."

Example:
Core Family = {Mother, Father, Children}.

Constraint — Concept *218

A constraint is a requirement that *explicitly* and *intentionally* tries to directly restrict any system or process. A key property of a constraint is that a penalty or loss of some kind applies if the constraint is not respected.
Constraints include limitations on the engineering process, a system's operation, or its lifecycle.

"A constraint is 'something that restricts'."
(The American Heritage Dictionary (Dell))

Notes:

1. There are two kinds of constraints: Binary and Scalar.
   Binary constraints are statements that tend to include the words 'must' or 'must not': that is, they tend to make demands about what is mandatory for the system or what is prohibited for the system. Binary constraints are declared either by using a Constraint parameter or by specifying 'Type: Constraint.'

   Example:
   C1: Constraint: A design idea must not be made of material, components or products only produced outside the European Market, if there is any EU material which can be used.
   C2: Constraint: The design must contain ideas based on our own patents whenever possible.

C3:
Type: Constraint.
Defined As: All stakeholder critical qualities must be planned and delivered so that they are viewed as obviously and significantly better than any competitor in the same price range.

From an attribute viewpoint, binary constraints are function constraints, design constraints or condition constraints.

Scalar constraints are specified for performance or resource attributes. They are specified using Fail and Survival parameters, which set the constraint levels on a scale of measure.

2. All engineering specifications (requirements and design) and management plans, once stated, potentially and probably have some constraining influence on the rest of the planning or engineering process. So all specifications and plans are 'constraints' in this sense.

   However, we can *clearly distinguish* between specifications where the *primary* intent is to constrain (like a mandatory constraint or a level for Survival), and those specifications where the primary idea is *not to constrain*, but to *motivate positively* (for example, a binary function target or, a scalar performance target, such as a Goal or Stretch level). We could classify the former as 'intentional and direct constraints' and the latter as 'unintentional and indirect constraints'.

   So, we only classify specifications and concepts as 'constraints' when the clear intent and primary purpose is to restrain, limit, restrict constrain or stop.
3. You have to look at constraints from a 'stakeholder' viewpoint. As with any requirement or design, what is a requirement for one level of system stakeholder, is a constraint as viewed by sub-ordinate stakeholder levels.

   One stakeholder's requirement is another stakeholder's constraint.
4. All constraints are valid for their associated defined conditions. Sometimes the constraint conditions are stated explicitly, sometimes they are implied or inherited from more global specifications.

   A 'global' constraint is usually imposed by higher levels of authority, or by earlier planning processes. For example: by company policy, law, contract, strategic planning or systems architecture.

   Example:
   An example of a global constraint:
   Availability Criteria: Constraint: No Company Product shall ever be designed with less than 99.5% availability.
   A corresponding local constraint setting a higher constraint level:
   Fail [US Market, Military Systems]: 99.98%.
5. Constraints can be classified by the 'relative level of organization' they apply to, as proposed by Ralph Keeney (1992):
   - Fundamental Constraints: handed down to us from higher authority
   - Strategic Constraints: ones we have imposed at our own level, over which we have control
   - Means Constraints: constraints imposed at levels supporting us, which we can therefore overrule.
6. All requirements, including all constraints, have different 'priority.' This priority (or 'power') is determined by the conditions (the

qualifiers) and by their related specifications (for example, by parameters like 'Authority'). It is a complex process to determine constraint priority, and the 'answer' is dynamically changing. There is probably no absolute constraint that must be respected 'no matter what.' That is, there might always be a higher priority consideration that overrides a given constraint. For example, 'Thou shalt not kill (except in self-defense).'

7. Constraints always represent, in some way, some of the values of some stakeholders. But a given constraint does not necessarily agree with the values of all stakeholders. The constraint of one stakeholder might be in direct conflict with the requirements of another stakeholder.
8. Any set of categories (including no categories!) can be specified for classifying constraints. System, performance, budget and design are an arbitrary few such categories.
9. I view constraints as borders around a problem. We can do anything we like within the borders, but we must not wander outside them.

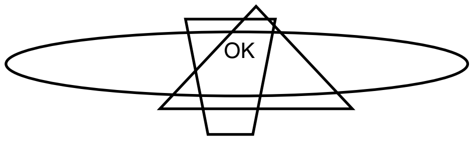


Figure G3
Constraints impose restrictions on both other requirements and designs. But the remaining space ('OK') gives considerable freedom to set more-exact requirements, and to specify more-exact designs.

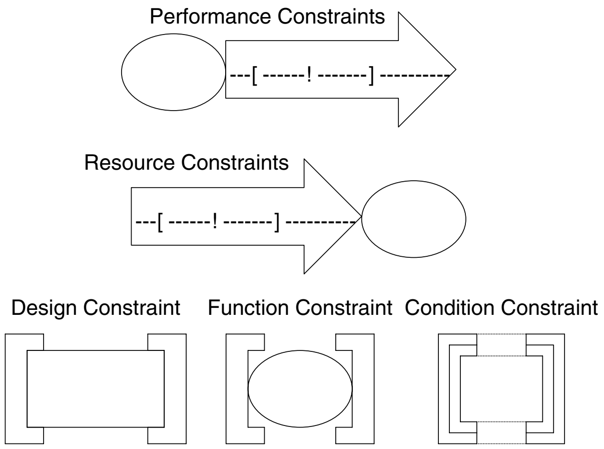


Figure G4
Drawn Icons for Constraints.

10. A requirement is a 'constraint on succeeding engineering processes' if and only if it has an authority, or other form of priority, which means that:
    - you must stay within its guidelines (when making later decisions or specifications)
    - you cannot change it yourself (in order to avoid obeying it), without authority to do so.

Related Concepts: Requirement *026; Function Constraint *469; Performance Constraint *438; Resource Constraint *478; Design Constraint *181; Condition Constraint *498; Survival *440; Fail *098.

Keyed Icons: For Survival: [and/or] and for Fail: !

"In context: ----[----!---]---->O---[-------!---]----->"

Continuous Process Improvement **Concept *424**

Continuous Process Improvement (CPI) includes any and all continuous long-term effort to systematically improve an organization's work processes.

Acronym: CPI *424.

Related Concepts: Defect Prevention Process (DPP) *042; Statistical Process Control (SPC) *466.

Core Specification **Concept *633**

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.

Example:
Core Specification Parameters include:
• Scale • Meter • Goal • Definition • Constraint

Notes:
1. Core specification can be distinguished from 'commentary' and 'background' (supporting) specification.
2. Core specification is the 'meat' in specifications of requirements, designs, Evo steps and test cases.

Synonyms: Implementable Specification *633.

Related Concepts: Non-Commentary *294; Background *507; Commentary *632; Specification *137; Specification Quality Control (SQC) *051.

Cost **Concept *033**

Cost is an *expense* incurred in building or maintaining a system. It is consumption of a resource.

Synonyms: Price *033; Expense *033: The degree of consumption, how much resource was used.

Related Concepts: Resource *199.

Keyed Icon: -|->O

"The keyed icon is a level symbol, '|', set on a resource scale of measure, '-->O'."

Note: The neutral symbol '|' is chosen to represent the generic cost concept, rather a currency symbol, because the resources involved are more than just money. If you want to link the icon to the idea of a cost range, then think of the chosen symbol as a minus sign, turned 90 degrees.

Example:
---||||||||-->O expresses a cost range.
The keyed icon for Cost [Money] is a currency symbol, default € (Euro).

Example:
--€-->O to express a money cost and ----<€€€€€€€€€€|=======>O to express a cost range.

Credibility **Concept *035**

Credibility expresses the strength of belief in and hence validity of, information. Within Impact Estimation, credibility is usually assessed for the evidence and sources supporting each specific impact estimate. Credibility is expressed as a numeric value on a range of credibility ratings from 1.0 (for perfect credibility) to 0.0 (for no credibility at all). These credibility values can be used to credibility-correct the impact estimates: each impact estimate is multiplied by its relevant credibility.

Example:
If an impact estimate were 40% and its credibility were 0.5, then the credibility-adjusted estimate would be 20% (40% multiplied by 0.5).

Critical Factor **Concept *036**

A critical factor is a scalar attribute level, a binary attribute or condition in a system, which can on its own, determine the success or failure of the system under specified conditions.

Notes:

1. A critical *failure* factor is usually specified as a constraint level (for example a 'Fail' or 'Survival' level), or as a binary constraint ('Constraint').
2. A critical *success* factor is usually specified as a target level (a 'Goal' or 'Budget' level), or as a binary target ('Target').

Related Concepts: Critical Success Factor *418; Critical Failure Factor *025.

DDP **Concept *041**

Acronym for Defect Detection Process *041

Defect Detection Process **Concept *041**

The Defect Detection Process (DDP) is part of Specification Quality Control (SQC), which also includes the Defect Prevention Process (DPP). It is the systematic, project-focused process of identifying specification defects.

Source: A detailed description of the DDP process can be found in (Gilb and Graham 1993) and (Wheeler, Brykcznski and Meeson 1996).

Rationale: This is to avoid the high cost of late defect removal (at test, or in field), or to avoid the high cost of the consequences of malfunctions caused by the defect: "A stitch in time saves nine."

Notes:

1. The DDP is 'project oriented' in that it is primarily concerned with a project's economics, rather than an organization's work process economics (that is the DPP concern).
2. DDP is not itself concerned with process improvement, but it provides a stream of data, concrete defect examples, and a working environment that can be used to feed into, and to help analyze effects of, a Defect Prevention Process.

Acronym: DDP *041.
Related Concepts: Specification Quality Control (SQC) *051; Defect Prevention Process (DPP) *042.

Defect Prevention Process **Concept *042**

The Defect Prevention Process (DPP) is a specific IBM-originated process of continuous process improvement. It is part of Specification Quality Control (SQC).

Notes:

1. The DPP process works towards continuous process improvement for ongoing, and especially future, projects in a larger organization. It is fed suggestions, and data, on problems, from the Defect Detection Process (DDP), and from other defect-identification sources, like testing and customer feedback.

   "An ounce of prevention is worth a pound of cure."

2. As reported by IBM, in organizations of 100 to 1,000 people, about 200 to 1,000 process changes may be implemented annually. On initial DPP implementation (the first project), 50% of the total number of historical defects may be eliminated in the first year of use and 70% eliminated within 2–3 years.

Sources:

- Inspired by classical Statistical Process Control ideas (Deming 1986), the Defect Prevention Process (DPP) was developed and refined (from 1983 onwards) by Carole Jones and Robert Mays of IBM Research Triangle Park NC with the aim of improving IBM's processes for software engineering, hardware engineering and administration (Mays 1995).
- A detailed description of DPP can be found in (Gilb and Graham 1993 Chapters 7 and 17).
- DPP was the direct inspiration for IBM assessment process Level 5 (Ron Radice cited in Mays 1995), US DoD Software Engineering Institute's Capability Maturity Model, CMM Level 5, and CMMI Level 5.

Acronym: DPP *042.
Related Concepts: Plan-Do-Study-Act Cycle (PDSA) *168; Specification Quality Control (SQC) *051; Defect Detection Process (DDP) *041; Continuous Process Improvement *424; Process Improvement *114; Process Improvement Suggestion *088; Process Meeting *119; Process Change Management Team *118.

Definition **Concept *044**

'Definition' or 'Defined As' is a parameter that is used to define a tagged term.

Notes:

1. The tagged term could then be re-used anywhere else within scope, and would always have this precise definition.
2. Any tagged statement is, in practice, a definition of that tag, so the use of the 'Defined' parameter is to make it *explicit* that the statement is intended as a reusable definition.

   Example:
   Trained: Defined As: At least 30 hours classroom, and 'passed' practical exam.
   Trained: At least 30 hours classroom, and 'passed' practical examination.

Trained: Def: At least 30 hours classroom, and 'passed' practical examination.
Trained: Definition: At least 30 hours classroom, and 'passed' practical examination.
Trained = At least 30 hours classroom, and 'passed' practical examination.
Equivalent definitions of the term, 'Trained.'

Example:
Reliability:
Scale: Hours to <complete> defined [Task: Default = Most Complex Task].
Fail [USA]: 5 hours. "Most Complex Task is suitable default for a Fail specification."
Goal [Europe, End Next Year]: 10 hours.
Most Complex Task: Defined As: The work task, which normally takes most employees longest clock time to complete on average.

Abbreviations: Def *044.
Synonyms: Defined *044; Defined As *044.
Keyed Icons: : *or* = "Whatever follows the icon symbol is a definition of the tag or parameter to the left of the symbol. '=' is less commonly used."

Dependency Concept *189

A 'dependency' is a reliance of some kind, of one set of components on another set of components.

Notes:

1. Any given component can be part of numerous dependencies (either having one or more dependencies, and/or having one or more dependencies placed on it).
2. The reliance involved in a dependency can be of many kinds. For example, there can be dependency for operation, for success, or for failure avoidance.
3. Qualifiers specify *implied* dependencies.
4. The parameter, 'Dependency,' or its synonym, 'Depends On,' is used to *explicitly* specify a dependency.

Example:
Z [T]: YY. "Only if T is true, does Z have a value of YY."
A: Depends On: B. "If B is not true then A is not true."
Tag A:
Dependency: XX. "Tag A has a dependency on XX."

Example:
Goal [Contract Beta]: 60%.
This means that the Goal level requirement of '60%' is valid as a Goal if, and only if, 'Contract Beta' is 'in force.' The Goal has an implied dependency *on the qualifier, 'Contract Beta.'*

Example:
Dependency: The satellite must be operational for the phone to operate. <- Catherine.

Example:
Dependency XX: Design Idea XX -> Reliability [USA, If Patent PP].
Example of dependency of an objective (Reliability [USA]), on both a design idea (Design Idea XX) and a condition (Patent PP). Note: -> = 'Impacts'.

This is a 'weak' dependency statement because we have no specification of whether the dependency is trivial or critical in degree. A numeric impact estimate could be used to give us that information, later, if we wanted it.

Example:
Tag: Refugee Transport.
Type: Function.
Description: Moving refugees back to home villages.
Source: Charity Aid Manual [March, Last Year].
Depends On: The mode of transport will be determined by safety and cost factors.
Or the equivalent:
Dependency: The mode of transport will be determined by safety and cost factors.

Example:
Contract Beta: Depends On: Conglomerate Corp [Our Customer, USA].
This means that if 'Conglomerate Corp [Our Customer, USA]' is not true, then 'Contract Beta' is not 'true.'

Rationale: To promote awareness of relationships, ensure more realistic planning, and provide the ability to identify reliance, and therefore cope with any associated risks.
Synonyms: Depends On *189.
Related Concepts: Before *312; After *313; Impacts *334; Is Impacted By *412.

Description — Concept *416

A description is a set of words and/or diagrams, which describe, and *partially* define, a component.
The parameter 'Description' is used to specify description.
Notes:
1. A description will convey the essence of a concept, or of a specification, but *the full definition* of the element may well require many other parameters to define it fully (from all interesting viewpoints), including implied or inherited definitions from other system components.
2. Models, real systems and prototypes can also provide a form of 'description.'

Example:
Mechanical Power:
Type: Function.
Assumption: At least 100 horsepower.
Constraint: Product Cost is lower than €500.
Risk: Last of European Commission Development Funding.
Version: March 1, This Year.
Description: The mechanical component that will provide all mechanical power to the system.
Note that the function description is only one part of the full function definition of 'Mechanical Power.'

Design Constraint — Concept *181

A design constraint is an explicit and direct restriction regarding the choice of design ideas. It either declares a design idea to be compulsory (Mandatory Design) or to be excluded (Prohibited Design).

Design constraints are dictated from an earlier system development stage (a higher level or a more specialized level). For example, the system architects pass on a number of design constraints, within the architecture specifications, to the system engineers.

A design constraint is a binary requirement. It can be a generic constraint or involve specific design(s).

Example:

================== Prohibited Designs ==================

P1: Constraint: Products and Services of direct competitors shall be avoided.

P2: Constraint: No software product version shall be released for sale until at least 3 month field trial has completed reporting no major faults outstanding <- Technical Director's Policy 6.9.

P3: Constraint [Europe]: No goods will be shipped without advance payment or bank guarantee.

================== Mandatory Designs ==================

M1: Constraint: Resident Workers in Country of Export shall be used wherever possible.

M2: Constraint [IT Projects, In House]: Commercial Off The Shelf Software shall be used exclusively.

M3: Constraint: Products and Services from Our Corporation, Our Customers and Partners are preferred <- Corporate Policy 5.4.

M4: Constraint [Programming]: Use Java as Programming Language.

Notes:

1. Some people use the term 'Design Constraint' to mean anything that constrains the choice of design. However, within Planguage the term is more restricted. It is a direct constraint on design ideas themselves; directly referring to design ideas, generically or specifically. All other types of requirements 'constrain' our choice of design, but not as directly as a design constraint.

   Indirect Constraints:

   - A Resource Constraint determines resource, and so impacts optional design.
   - A Performance Constraint determines performance, and so impacts optional design.
   - A Function Constraint determines function, and so impacts optional design.
   - A Condition Constraint determines conditions, and so impacts optional design.

   Direct Constraints:

   - Design Constraints determine design directly, by specifying a mandatory design or a prohibited design.

     All requirement types: targets and constraints – have some potential effect on our design choices. But, design constraints are 'direct' in the sense that they make specific design decisions.

     Example:

     Spruce Goose:

     Type: Generic Design Constraint.

     Definition [If Wartime]: A troop transport plane may not use scarce <metal alloys>.

Howard Hughes' airplane, 'The Spruce Goose,' had this design constraint before the end of the Second World War. He made the plane largely of 'spruce' wood.

2. Only designs that are 'design constraints' should be allowed within *requirement* specifications. All *optional* design ideas, designs you can swap out if you find a better one, should be specified in *design* specifications. This is so that each level of design responsibility knows what it is free to do, and not free to do.

Synonyms: Architectural Constraint *181; Design Restriction *181; Constrained Design: Informal Use; Required Design: Informal Use; Solution Constraint: Informal Use.

Related Concepts: Constraint *218; Requirement *026; Design Idea *047; Condition Constraint *498.

Design Engineering — Concept *501

Design Engineering is an iterative process of determining a set of designs, with rigorous attention to quantified and measurable control of their impact on requirements.

The design engineering process implies the matching of potential and specified design ideas with quantified performance requirements, quantified resource requirements, and defined design and condition constraints.

Notes:

1. Planguage involves design engineering. By contrast, conventional 'design' activity (the kind of 'design' often found in the literature and in practice) usually has a less systematic, less quantified process, using perhaps intuition, tradition, and more trial and error, to determine satisfactory technology and to determine stakeholder satisfaction. It is characterized by naming objectives (for example, 'better usability'), and naming designs (for example, 'single standard interface'), but not following up with *quantified* versions (that is, providing the information captured in Planguage, using such parameters as Scale, Goal, Impact Estimate and Meter).

Related Concepts: Design Process *046; Architecture Engineering *499; Systems Engineering *223; Engineering *224.

Design Idea — Concept *047

A design idea is anything that will satisfy some requirements. A *set* of design ideas is usually needed to solve a 'design problem.'

Notes:

1. A design idea is not usually a requirement. However, a design idea can be a requirement if it is a *design constraint.* That is, a specific design is stated as mandatory or prohibited in the requirements.
2. Requirements are inputs into a design process; design ideas are the outputs.
3. A design idea can, in principle, be changed at any time for a 'better' design idea (without having to ask the permission of any stakeholders because the system designers are responsible for the proposed design ideas). A 'better' design meets the requirements by giving more performance and/or less cost.
4. A satisfactory design idea can have some negative performance scalar impacts, and still be acceptable overall. As long as the negative impacts (negative side effects) of a design idea do not prevent us from reaching all the required target levels, the design idea can be used.

Figure G5
The drawn icon for a Design Idea *047.

5. A design specification is a written definition of a specific design idea. (See also the design specification template.)

Synonyms: Design *047; Strategy *047; Proposed Solution *047; Means *047.

Related Concepts: Architecture *192; Policy *111; Design Constraint *181; Design Specification *586; Design Problem *048.

Drawn Icon: A lying-down rectangle. (The standing rectangle is a document icon.)

Design Process — Concept *046

The design process is the act of searching for, specifying, evaluating and selecting design ideas, in an attempt to satisfy specified stakeholder requirements.

Design is finding a set of solutions (design ideas) for a set of defined requirements.

Overview of the Design Process:

- Analyze the Requirements
- Find and Specify Design Ideas
- Evaluate the Design Ideas
- Select Design Ideas and Produce Evo Plan

Design can be carried out in several ways. It can be based on tradition, on intuition, on dogma, on principles or heuristics. It can also be based on multidimensional quantified logic – this latter we would call 'engineering' or 'systems engineering.'

> "Design comes about entirely from the playing out of the evolutionary algorithm." <-*Susan Blackmore.*[1]

Related Concepts: Design Engineering *501; Systems Engineering *223; Engineering *224.

Design Specification — Concept *586

A design specification is the written specification of a design idea. A set of design specifications is the main output of a design engineering process. A specific set of design specifications, when implemented, will, to some degree, meet the stated requirements.

Notes:

1. A set of design specifications attempts to solve a design problem. Identification and documentation of the individual design ideas, and their potential contribution towards meeting the requirements, helps selection of the 'best' design ideas for implementation.

[1] Blackmore, Susan, *The Meme Machine*, Oxford: Oxford Paperbacks, 2000, ISBN: 019286212X. See Page 205.

2. The design engineering process uses the requirement specification as input. The design engineering process output is a set of design (solution) specifications (of design ideas).
3. See the design specification template for details of the required specification data.
4. The design specifications might contain information about the *expected* attributes of the designs for meeting requirements. This 'expected attributes' information of a design specification might be in the form of an Impact Estimation table or, it can be as simple as an assertion of impacts on requirements, referenced by their tags (see example below).

Example:
Engineer Motivation:
Gist: Motivate, using free time off.
Type: Design Idea.
Impacts [Objectives]: {Engineering Productivity, Engineering Costs}.
Impacts [Costs]: {Staff Costs, Available Engineering Hours}.
Definition: Offer all Engineers up to 20% of their Normal Working Hours per year as discretionary time off to invest in Health, Family and Knowledge {Studies, Write Papers, Go to Conferences}.
Source: Productivity Committee Report 1.4.3.
Implementor: Human Resources Director.

Template: Design Specification Template.
Abbreviations: Design Spec *586.
Synonyms: Technical Design: Informal use; 'The Design': Informal use.
Related Concepts: Design Engineering *501; Design Idea *047; Systems Architecture *564; Architecture *192; Architecture Specification *617; Specification *137.

Deviation **Concept *475**

Deviation is the amount (estimated or actual) by which some attribute differs from some specific benchmark or target. Deviation is usually expressed numerically using either absolute or percentage difference.
Synonyms: Variance *475.
Keyed Icon: ±

DPP **Concept *042**

Acronym for Defect Prevention Process *042

Due **Concept *554**

'Due' is a parameter indicating when some aspect of a specification is due.

Example:
Due [Sample A]: End of January Next Year <- Contract Section 3.5.6 [Supplier X].

Synonyms: Deadline *554; Due Date *554.
Historical Note: The idea of 'Due' as a parameter was from an unpublished note by Jens Weber, Daimler Chrysler, Frankfurt.

During **Concept *314**

'During' is used when specifying events (including Evo steps and tasks) to indicate a time dependency for events that must be carried out concurrently (that is, done in parallel).

Example:
Step 33: Step: {A During B During C}. "Do A, B and C concurrently."

Elementary — Concept *055

An 'elementary' component is not decomposed into sub-components.

Notes:

1. A component can be elementary because it is unable to be decomposed into sub-components, or because there is no declared intent to decompose it.
2. The decision to subdivide a complex concept into elementary concepts is a practical and economic matter. It depends on:
   - the size and complexity of a project
   - the need for precise control over system attributes
   - the risks taken if specification detail is inadequate
   - engineering culture
   - intellectual ability to decompose
   - other factors.

   Even when an initial decision is made about having no further decomposition of an idea, later events and opportunities, or later more detailed phases of systems engineering, may cause a concept to be decomposed into elementary concepts.

   The reverse can be true too. Initial decomposition may seem unnecessarily detailed or unnecessarily constraining. So, the concept may be simplified from a complex concept back to an elementary concept.
3. The essential characteristic of scalar attributes, which tells us if they are elementary, is the number of defined scales of measure. There is *only one* distinct Scale for each elementary concept.
4. Elementary concepts are directly measurable or testable. You can only test or measure a complex concept by way of testing and measuring the set of its elementary concepts. A complex concept is not the 'sum,' but the 'set' of its elementary concepts.
5. Normally an elementary statement can have its own distinct 'tag,' and can be treated (developed, tested, costed, quality controlled) relatively independently of any other elementary statement.

Related Concepts: Complex *021.

Error — Concept *274

An 'error' is something done incorrectly by a human being.

Notes:

1. Errors are usually committed unintentionally; they are often forced to happen by 'bad' work processes (Statistical Process Control Theory (Deming 1986; Juran 1964; 1974)).
2. Human errors in specification processes lead to defects in specification or evaluations. For example, errors in systems engineering processes result in (written) engineering and contractual specification defects. In turn, specification defects result in faults in the system, which may or may not, result in system malfunctions (the fault actually occurs). See related concepts.

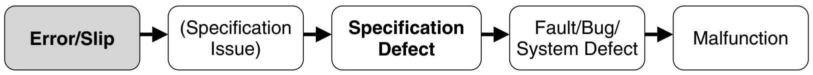


Figure G6

3. SQC is a means of checking specifications to discover whether errors have been made. Any suspected violation of any applicable specification rule is logged as an issue.

"To err is human."
Saying, and similar to Plutarch (AD 46–120) "For to err in opinion, though it be not part of wise men, is at least human".

Synonyms: Slip *274.
Related Concepts: Specification Issue *529; Specification Defect *043; Fault *339; Malfunction *275.

Estimate — Concept *058

An estimate is a numeric judgment about a future, present or past level of a scalar system attribute. (This includes all performance and cost attributes.)
Notes:
1. Estimates are usually made where direct measurement is:
   - impossible (future), or
   - impractical (past), or
   - uneconomic (current levels).
2. An estimate is usually extrapolated from available information, and past experience.
3. An estimate can be made for numeric facts from the *past* (benchmarks), even if precise past data is not available.
4. Estimates are made about any scalar system attribute: cost levels, resource availability, quality levels, savings and other dimensions of systems.

Estimate, To — Concept *059

In Planguage, to 'estimate' is:
The process of arriving at a judgment by guessing the probable numeric value of a numeric attribute level using other methods than immediate measurement.

Estimate: "to judge or determine generally but carefully (size, value, cost, requirements, etc.); calculate approximately."
Webster's New World Dictionary

Estimation is not to be confused with Quantification or Measurement. (See figure in concept, 'Quantify, To *385'.)
Related Concepts: Estimate *058; Quantify, To *385; Specify, To *239; Measure, To *386.

Event — Concept *062

An event is a specified occurrence.
Example:
- President Inaugurated
- Process Begun
- Process Ended
- Task Started
- Task Interrupted
- Contract Signed

Notes:
1. 'Event' is not used here in the 'organized occasion' sense of the term. In other words, it is not used in the sense of a 'wedding' or a 'gala opening of a building' being an 'event.'

2. An event is not the 'carrying out' of an activity, such as performing a task, a process, or some systems engineering implementation (like implementing an Evo step).
3. An event must be clearly distinguishable from the non-occurrence of the event. It must be reliably observable, testable or measurable in the real world. Consequently, events must be precisely defined – unambiguously.

   However, the occurrence of an event is not the same as our measurement of it. An event occurs whether or not it is immediately detected or measured.
4. An event usually results in some measurable change in a status. If a specific event has occurred, then any status associated with the event will have changed. By evaluating the relevant event condition, the setting of a status can be determined.
5. An 'event condition' can be defined as a qualifier condition – the event must have happened for the qualifier condition to be true.

   Example:
   Goal [First Sale]: 9% Or Better.
   First Sale: Event: We make our first sale of refrigerators to the USA.
   Past [Last Year, Europe, First Flight]: 98%.
   First Flight: Type: Event. Description: Successful first flight <officially logged>.
6. To attempt a more detailed definition:[2] An event is an occurrence (a set of circumstances, which include changes in status), localized in space and time, which results from some activity and which is significant as an indicator of progress or as a stimulus (acts as a trigger) for other activity.
7. An event can be defined in time and space (theory of relativity), but it can be conceived of, specified and defined without time and space coordinates.

Synonyms: Happening; Occurrence; Point (in space-time).
Related Concepts: Qualifier *124; Condition *024; Status *174.

Evidence — Concept *063

Evidence is the historic facts, which support an assertion. The evidence usually will have been the basis for making an assertion.
In Impact Estimation, evidence is required for each impact estimate. Where there is no evidence, it should be clearly stated that there is none.

Example:
Design B -> Goal 1.
Scale Impact: 10 minutes.
Evidence: Of 100 surveyed Customers last year, 30 agreed there was this level of impact on Goal 1 <- Marketing Report A123.

Evo — Concept *355

Abbreviation for Evolutionary Project Management *355 and Evolutionary *196.
Readers will have to bear with me that I use this abbreviation for both 'Evolutionary Project Management' and 'Evolutionary.' The underlying concept is the same <-TG.

[2] With thanks to Don Mills, New Zealand.

Evo Plan **Concept *322**

An Evo plan is a set of sequenced and/or a set of yet-to-be sequenced Evo steps. The current planned sequence of delivery of any of the steps should be reconsidered after each Evo step has actually been delivered and the feedback has been analyzed. Many factors, internal and external, can cause a re-sequencing of steps and/or the insertion of previously unplanned additional step(s) and/or the deletion of some step(s).

It is the identification of the *next* Evo step for delivery that should be our focus for detailed practical planning. After all, at an extreme, the other planned steps may never be implemented in practice.

Notes:

1. For the Evo steps, an Evo plan is likely to specify only the tag names. Detailed specification of an Evo step will be held in its step specification.
2. An Impact Estimation table may be included in an Evo plan. Such a table would hold the estimates for the impacts of each of the Evo steps, and the feedback after delivery of each of the Evo steps. The progress made could then be tracked against the Evo plan estimates.

Synonyms: Evolutionary Plan *322; Evolutionary Delivery Plan *322.

Related Concepts: Evolutionary Project Management (Evo) *355; Evolutionary *196.

Drawn Icon: A series of any number of steps, each one representing an Evo step.

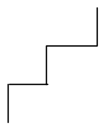


Figure G7
The drawn icon for Evo Plan *322 consisting of several Evo Steps *141.

Evo Step **Concept *141**

An Evo step ('evolutionary step' or simply 'step') is a 'package of change,' containing a set of design ideas that on delivery to a system is intended to help move the system towards meeting the yet-unfulfilled system requirements.

Evo steps are assumed to be small increments, typically a week in duration or 2% of total budget. There are two purposes for Evo steps: to move us towards the long-range requirements, and to learn early from stakeholder experience (with a view to changing plans and designs early).

Notes:

1. Evo Step Content: A step will contain the 'means' for meeting specified 'ends' (requirements). It will contain some combination of design ideas, which aim to achieve the requirements.
2. Dynamic Step Sequencing: Evo is conceptually based on the Plan-Do-Study-Act cycle. An Evo plan for a project consists of a planned series of Evo steps sequenced in order for delivery. Step sequencing for delivery can be roughly sketched or planned in actual time sequence. Step

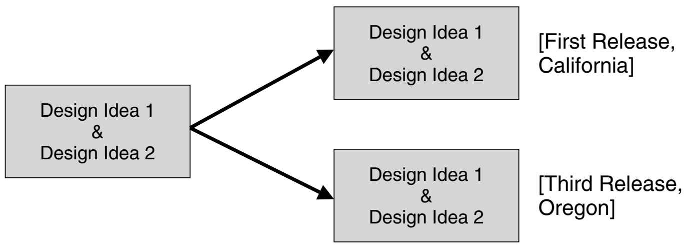

Design Idea 1 & Design Idea 2
Design Idea 1 & Design Idea 2
[First Release, California]
Design Idea 1 & Design Idea 2
[Third Release, Oregon]

Figure G8
Evo Step packaging. Delivery-specified Evo Steps: same underlying step specification, but two different times and places.

sequencing always remains finally contingent upon actual feedback results, and on external considerations such as new requirements, changing priorities or new technology. The step delivery sequence is determined dynamically, after the previous step results are analyzed (Study Phase) and a decision is made about the next step (Act Phase). Selecting the *next* step for delivery is the main focus of Evo planning activity.

3. Step Priority: Steps with the highest stakeholder value to cost ratios or performance to cost ratios ought to be scheduled for early delivery. Step dependencies have also to be considered.
4. Step Size: A step is typically, but not unconditionally, constrained to be between 2% and 5% of a project's total financial budget and total elapsed time. Why? Well, because 2% to 5% is a reasonable amount of resource to gamble, if you are not absolutely sure whether a step will succeed.
5. Step Specification: An evolutionary step specification is the written description of the step content; that is, specification of the list of design ideas involved.
6. Step Lifecycle Location: A step is developed and delivered within a result cycle: any necessary step development occurs as part of the development cycle, any step production required occurs as part of the production cycle, and step delivery takes place as part of the delivery cycle.
7. Step Content Reuse: It is possible for the same step *content* to be repeated in several *different* steps (that is, in effect a 'roll-out' across a system, over time, such as to different countries or states or branch offices). In such cases, the step specifications will differ only in the qualifiers. For example, Step 1: Function XX [California], Step 2: Function XX [New York], and so on.

   Example:
   S23: Step [<Time, Place, Event to be determined>]: F1, F2 [Europe], D3 [China].
8. The main difference between an Evo step package (above example with undetermined qualifier) and a delivery-specified Evo step is that the latter has been assigned a sequence or timing and a place of application qualifier. The Evo step package is just a specification of the step contents. An Evo step package can be deployed in multiple times and places. You could say that an Evo step package is a reusable specification. For example, every week it could be to different countries.

Synonyms: Step *141; Evolutionary Step *141; Build, Increment, Installment, Release: Near synonyms depending on whether there is use of feedback and dynamic change (Royce 1998).

Related Concepts: Evo Step Specification *370; Evo Plan *322; Evolutionary *196; Evolutionary Project Management (Evo) *355; PDSA Cycle *168: See also the individual Plan, Do, Study, Act components; Result Cycle *122; Development Cycle *413; Production Cycle *407; Delivery Cycle *049; Result *130: Synonym is Step Result.

Keyed Icon: ->☺ or ->:) "Symbolizing 'Impact' on stakeholder. The ☺ symbol is sometimes automatic 'correction' for the colon and right parenthesis keyed symbols, in Microsoft Word."

Evolutionary — Concept *196

The 'evolutionary' concept implies association with Evolutionary Project Management; an iterative process of change, feedback, learning and consequent change. Evolutionary processes needs to be carefully distinguished from other processes – those that do not iterate, do not learn from experience, and do not cater for change.

Abbreviation: Evo *196.

Related Concepts: Evolutionary Project Management *355; Evo Plan *322; Evo Step *141; Plan-Do-Study-Act Cycle *168.

Evolutionary Project Management — Concept *355

A project management process delivering evolutionary 'high-value-first' progress towards the desired goals, and seeking to obtain, and use, realistic, early feedback.

Key components include:

- frequent delivery of system changes (steps)
- steps delivered to stakeholders for real use
- feedback obtained from stakeholders to determine *next* step(s)
- the existing system is used as the initial system base
- small steps (ideally between 2%–5% of total project financial cost and time)
- steps with highest value and benefit to cost ratios given highest priority for delivery
- feedback used 'immediately' to modify future plans and requirements and, also to decide on the next step
- total systems approach ('anything that helps')
- results-orientation ('delivering the results' is prime concern).

Description: Chapter 10, "Evolutionary Project Management: How to Manage Project Benefits and Costs".

Abbreviation: Evo *355.

Synonyms: Evo Management *355; Evolutionary Delivery Management *355; Rapid Delivery Management (Acronym: RDM), Result Delivery (These synonyms are used within Jet Propulsion Labs. (Spuck 1993); Synch-and-stabilize or Milestone Approach (These synonyms are used within Microsoft (Cusumano and Selby 1995); None of them are perfect synonyms, but since each author and company has a long list of extremely similar features that make up these processes, they are close enough.

Historical Note: Evolutionary Project Management had an early large-scale documented use in the Cleanroom techniques used by Harlan Mills within IBM in the 1970s (Mills 1980). Larman and Basili (2003) gives a comprehensive history of the method.

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

Except **Concept *389**

'Except' is used to specify that the following term or expression is an exception from the previous term or expression.

Example:
Goal [Europe Except {Denmark, France, Luxembourg}]: 20%.

Fail **Concept *098**

'Failure' signals an undesirable and unacceptable system state. A Fail level specifies a point at which a system or attribute failure state begins. A single specified number (like Fail: 90%) is the leading edge of a Failure Range.

A Fail parameter is used to specify a Fail level constraint; it sets up a failure *condition*.

Notes:

1. Failure ranges can be arbitrarily stipulated by a stakeholder. They might be stated in a contract. They are specified so as to keep designers and implementers aware of the levels at which stakeholders are likely to experience failure, or to contractually declare some degree of failure.
2. A failure *range* maps an extent of unacceptable levels. The failure range is better than 'catastrophic' levels, and worse than 'acceptable' levels. In other words, the failure range extends from the defined Fail level in the direction of 'worse' until a Survival level (or Catastrophe level) is reached.
3. The purpose of the 'Fail' concept is to inform us that we need both to design for, and operate at, more acceptable levels.

   Example:
   Fail [Euro Release]: 99.5%.

   For example, a state of failure can result from issues such as safety problems, operator discomfort, customer discomfort, loss of value, and loss of market share. Failure levels cause problems, even temporary system loss, but they are not immediately critical to a system's continued survival. The assumption is that it is possible to get the system out of a failure range.
4. Fail levels do not represent *total* failure. That role is defined by catastrophe levels. However, system development should keep going until, at least, the actual system levels are better than the specified failure levels. Otherwise, they are delivering some degree of failure to some stakeholder; that is, the system or attribute will at some stage fail in some sense.
5. A Fail constraint specification means that some defined stakeholder has stated the level at which the attribute's numeric value becomes unacceptable to them. Any level equal to or worse than the Fail level, is outside the 'acceptable' range for that stakeholder.
6. A systems engineer should document *why* a specific Fail level was chosen (using Rationale or similar), and the likely impacts (using Impacts) and consequences of any failure (using Risks), so that risk analysis and prioritization can be carried out.

   Example:
   Learning Time:
   Scale: Mean Time to Learn defined [Task] by defined [Operator].
   Fail [Outgoing Call, Beginner]: 3 minutes <- Marketing Requirement 3.4.5.

Risk: If the Mean Time is not lower, then Competitor Products will be perceived as better and we will lose <market share> <- Marketing Planner [Andersen].
Fail [Address List Update, Professional User]: 30 seconds <- Marketing Requirement 3.4.6.
Authority: External Consultants. "Outside consultants tell us we will be rated badly if we fail to beat this level."
Goal [Average Task, Average User]: 25 seconds <- Marketing Requirement 3.4.7.
Rationale: Marketing believes this will make us best in the Market.
The local parameters, Risk, Authority and Rationale can be used to explain why scalar levels have been set at specific levels. Note that the Source(s) of information (format: 'B <- Source of B') give indirect authority for the specification levels. (The Goal specification is included here to give a more realistic specification example.)

Synonyms: Fail Level *098; Fail Limit *098; Failure Level *098; Failure *098; Warning *098; Must (Avoid) *098: Historical usage only.
Related Concepts: Survival *440; Catastrophe *602; Range *552: See 'Failure Range'; Must Do *539: Historical usage only.
Keyed Icon: ! "In context on scalar arrows: ---!--->O---!--->
A Failure Range would use multiple Fail icons: ----!!!!!!--->-> "

Frontroom — Concept *343

Frontroom is an adjective or noun, referring to a conceptual place, used to describe any project management processes or activities, in Evo, that are visible to the Evo step recipients.
Notes:
1. Typically, 'frontroom' is used to refer to the delivery cycle part of the result cycle. The frontroom is where the step is delivered to the stakeholders.
2. The frontroom is where stakeholder-level results of the step integration can be tested and measured.

Related Concepts: Backroom *342.

Function — Concept *069

A function is '*what*' a system does.
A function is a binary concept, and is always expressed in action ('to do') terms (for example, 'to span a gap' and 'to manage a process').
Notes:
1. A function has a corresponding implied purpose. For example 'to span a gap' usually has as an implied purpose to enable something to get from point A to point B over the gap.
2. Function is a fundamental part of a system description: a system consists of function attributes, performance attributes, resource (cost) attributes and design attributes. All attributes exist with respect to defined specified conditions.
3. All the system attributes must be described together, in order to fully understand a real world system. Function is a 'pure' concept, which cannot exist in the real world alone. Functions need to have associated with them, the relevant performance and resource attributes.

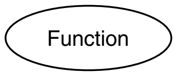


Figure G9
The drawn icon for Function *069.

Further, functions can only exist and successfully interact with the real world if they have certain minimal levels of such attributes, for example, levels of availability.

4. Function is a system attribute that is expressed without regard to the related performance, cost and design. A function needs to be clearly distinguished from design ideas. Design ideas are a real world method for delivering all the function, performance and cost attributes of a system.
5. Function itself is binary. Function in a system is either implemented or not. It can be tested as present or not. Any *real* implemented function will have some associated performance and cost attributes – whether we planned for them or not. Once a design with the required functionality is specified (or later implemented), we need to consider whether that particular design has *satisfactory* performance and resource (cost) attributes. In other words, we control these scalar attribute's levels mainly by specifying appropriate design options, which deliver the required performance and cost levels.
6. A function can often be decomposed into a hierarchical set of subfunctions. For specification clarity, prefixes such as 'sub-', 'supra-' or 'family' relationships (such as kid, parent, sibling) can be used to express the relationships amongst the different functions. Alternatively, the parameters, Includes, Is Part Of and Consists Of can be used.
7. I have intentionally chosen the term 'function' as the adjective for 'function' (for example, in 'function specification' and 'function requirement'), rather than the more common 'functional.' It is the 'requirement for function' that is being expressed, rather than 'making the requirements functional.' The logic of this choice is the same as for choosing 'quality' (for example, in 'quality requirements), rather than 'qualitative.'

Related Concepts: Attribute *003; Design Idea *047; Mission *097; Function Design *521.

Drawn Icon: An oval (a circle would also be considered a function.)

Keyed Icon: O or parentheses, ( ) "In context: ------->O------> This describes a system: the function keyed icon, 'O,' is combined with two scalar arrows representing scales of measure for cost and performance attributes. Alternatively: ----->(<*function tag*>)----> "

Function Constraint — Concept *469

A function constraint is a requirement, which places a restriction on the functionality that may exist in a system.

A function constraint is binary: it specifies that a specific function must be, or must not be, present. The implication is that some kind of failure will result if a function constraint is not met (such as contract penalties).

Example:
No New Games:
Type: Function Constraint.

Rationale: No new games of any kind will be available on the new product.
Definition: No functionality required solely for a New Game is to be developed.
Support for Old Games [Release 1]:
Type: Function Constraint.
Rationale: All available games from our older product will be available to any customer on request. Some customers would be upset at losing the existing games.
Definition: Functionality to support Old Games must be included.
Related Concepts: Function *069; Function Target *420; Requirement *026.

Function Design **Concept *521**

Function design is a design primarily aimed at satisfying specified function requirements. A specified function design has two characteristics, which we primarily select it for:

- function requirement satisfaction, and
- satisfactory consequent levels of performance and cost attributes.

Example:
Cross River:
Type: Function Requirement.
Definition: Move people and goods from one shore to the opposite shore of a river.
Function Design Ideas [Version 1]: {Build a Bridge, Use a Boat, Swim Over "minimal design," Take Route 'Around' the River, Fly Over}.
Consideration of potential design ideas: Function Design Ideas [Version 1] shows selecting on function satisfaction: some function designs, which satisfy the function requirement.

Example:
Function Design Ideas [Version 2]: {{Build a Bridge and/or Use a Boat}, Not {Swim Over, Take Route 'Around' the River or Fly Over}}.
Function Design Ideas [Version 2] shows further selection using knowledge of performance and resource (cost) attributes: The function designs that look most promising for the system.

Notes:

1. The final real performance and cost levels delivered is dependent on the *specific* design chosen (for example, exactly what specific *design* of bridge, or specific *type* of boat).
   Functional design necessarily narrows the *remaining* design scope to some degree. It can even narrow the design scope to a set of function designs *without* actually taking a *final* choice of specific design (for example, Build a Bridge and/or Use a Boat – without yet saying exactly which type). The final design specification would then be left to a downstream design process.
   This delay might be justified by their more specialized knowledge downstream, or justified by the advantage of putting off the decision due to changed technology/market conditions/costs, or due to an advantage of making the decision in the context of many other system/project-wide decisions (avoiding sub-optimization).

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

2. Any function design in its real implementation (as opposed to pure function specification) will impact many of our non-function requirements (performance requirements, resource (cost) requirements, condition constraints or design constraints). This multiple impact is inevitable whether we like it or not. We cannot, it seems, only design for one pure requirement dimension without having some effects on the others.

   When a design is primarily specified for *non-function* purposes (like improving a quality level), it might inadvertently impact existing functionality as a side effect. This might possibly be acceptable. It might introduce new function, modify old function or make existing function inaccessible totally or practically.
3. The key reasons for considering function design, as a distinct design type, is that:
   - you can narrow the function design scope gradually
   - you become more conscious of the side effects on performance and cost
   - you become more conscious of the necessity of choosing function design alternatives on the basis of their impacts on performance and cost.
   - you can separate the design rationale for the function, from consideration of the other attributes.

Related Concepts: Function *069; Function Requirement *074; Design Process *046; Design Specification *586.

Function Requirement — Concept *074

A function requirement specifies that the presence or absence of a defined function is required.

A function requirement is binary, and can either be a specific function target or a generic function constraint.

Example:

Voice Recognition:

Type: Function Requirement.

Definition: The ability to recognize a human voice in terms of vocabulary and individual voiceprints.

Step 1: Step: Voice Recognition [Europe, If Company C has this function on the market].

Voice Recognition is defined as a function. It is then 'required' to be delivered in Evo step 'Step 1,' only in 'Europe' and only if 'Company C has this function on the market.' A specific design to implement Voice Recognition needs to be specified.

Notes:

1. Do not include technical design ideas in function requirements. Designs are quite different from functions. If designs are mandatory, then they should be specified as design constraints. A function is an abstract concept specifying activity of some kind, which is implemented by a design. For example: An accounting application (a design) provides a solution to support Maintaining Accountancy Information (a defined function).
2. Distinction should be made between a function target and a function constraint. A function constraint implies that a function must be

present or absent (subject to its qualifier conditions) in a system, or a penalty of some kind will be incurred.

3. By default, if there is no information that a function requirement is actually a function constraint, or 'Type: Function Constraint' specification, a function requirement is assumed to be a function target.
4. To authority levels, which are lower than the one that specified it, a function target does become mandatory. If a lower authority disagrees with a requirement they have to take the issue up with the higher authority.
5. A function requirement is satisfied by any design, which meets the function description. For example, {transport via a bridge, transport by air, transport across water} all meet the function requirement 'to transport people from shore to shore of a river.' Note, in this example, the designs are high-level and are actually functions. They can be termed 'function designs.' A lower, more specific level of design {by public transport over a bridge, by hot-air balloon, by canoe} can also be considered.

   At the early rough stages of design, function requirements are best satisfied by rough function designs (like 'bridging the river'). At the latter stages of design, specific designs are better, like 'rope bridge.' The issue is that the more specific a design is, the less freedom of design choice remains, but the greater the knowledge of its attached performance and resource attributes. For example, the quality attributes of a software package selected to satisfy the function requirement, like reliability and portability.

Synonyms: Functional Requirement *074.

Related Concepts Function Target *420; Function Constraint *469; Function Design *521; Function *069.

Function Target — Concept *420

A function target is a specified function requirement. We need to plan delivery of the function under the specified conditions.

A function target can be contrasted with the other class of function requirement, a function constraint. A function constraint specifies mandatory functionality (either a function has to be present or absent), as a penalty of some kind will result if the constraint is not met.

Example:
Propulsion Capability:
Type: Function Target. "Could also be termed a Function Requirement."
Description: A means to mechanically drive the vehicle around in three dimensions.
Basic definition of a function target.

Example:
Step 22:
Type: Evo Step.
Dependency: Step 23 completed successfully.
Step Content: Propulsion Capability [Version = Prototype, Capability = Surface Movement, Means = Electrical-Powered].
Exploitation of the function target specification by referencing its tag in an Evo step plan, with suitable qualifiers. Notice how the qualifiers make the generic function somewhat more specific.

Related Concepts: Function Constraint *469; Function Requirement *074; Target *048; Function *069.

Fuzzy **Concept *080**

A specification, which is known to be somewhat unclear, potentially incorrect or incomplete, is called 'fuzzy.' It should be clearly declared as 'fuzzy.'

The keyed icons '< >' are used to explicitly mark any fuzzy specifications.

Example:

Scale: <define units of measure>.

Note: This is a template with a hint in fuzzy brackets.

Goal [<Europe>, <2005>]: <66%>.

Rationale: The idea is to avoid forgetting to improve specifications, and to avoid misleading other people into thinking you have done your potential best, when you know better should be done, when you have time and information, in order to define specifications at the necessary quality level.

Notes:

1. The obligation to mark dubious specifications with fuzzy brackets is typically adopted as a generic specification rule.
2. In general, a fuzzy term or expression should be enclosed in <fuzzy brackets>, but alternative notations (such as '??') can also be used.
3. Fuzzy brackets are used in electronic templates to indicate something to be filled out, and usually to give a hint as to what should be filled out. See Scale in example above.
4. A fuzzy specification essentially amounts to a declaration by the writer that the specification is defective at that point.

Keyed Icon: <*fuzzy term*>

Gap **Concept *359**

For a scalar attribute, a gap is the range from either:

- an impact estimate, or a specific benchmark (usually the current level),
- to a specific target (or occasionally, to a specific constraint).

Notes:

1. In general, the larger the gap, then the greater the need to deal with it ('the higher the priority') in order to reach the target or constraint. Of course, large gaps could be easy and some small gaps could be difficult, so that is why this paragraph says ' in general.'
2. When a gap no longer exists for a specific scalar attribute, then that attribute ceases to have 'claim on project resources' (priority). It then has no priority.

Related Concepts: Range *552; Design Problem *048.

Gist **Concept *157**

A Gist parameter is used to state the essence, or main point, of a specification. A Gist is a summary of the detailed specification.

Notes:

1. A good Gist serves two purposes:
   - it helps a planning group to agree on the summary of a specification, before they spend more time formulating the specification in greater detail.
   - it summarizes a detailed specification. This serves several purposes:
     – readers can quickly grasp the subject matter
     – readers can decide to avoid the detail

– presenters can refer to a specification by using just its tag name and Gist. (A reader who needs more detail can 'drill down' to the detail using the tag name.)

Example:
Gist: All the functions related to transportation of people.

Example:
Software Interfaces:
Type: Architecture.
Σ: The total set of software interfaces needed for this project.

2. The detailed specification may already exist, or it may be made on the basis of an agreement about the Gist.
3. When summarizing a *scalar* specification, use the more specific parameter 'Ambition.'

Synonyms: Summary *157.
Related Concepts: Ambition *423.
Keyed Icon: Σ "Greek Summa, mathematical summary symbol."

Goal — Concept *109

A goal is a primary numeric target level of performance. 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.

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].

Notes:

1. To reach a Goal level is a *success* to specific stakeholders. It is also a sort of 'stop' signal (a red light) for use of project resources on the specific performance attribute concerned: although better levels might be reached, and might be of value to some, they are not called for, under the stated conditions. For example, the additional value gained, given the estimated costs, is not viewed as worthwhile. In economic terms, we have at the Goal level probably reached the point of diminishing return on investment.
2. 'Goal' is intentionally not used for *resource* targets ('Budget' is used instead).
3. I now prefer the term 'Goal' instead of my traditional 'Plan' parameter. 'Plan' refers to so many other elements of planning. If an alternative were needed for Goal, I would use the more explicit 'Planned Level.'

Example:
Glory: "Humpty's and Alice's problem, what does 'glory' mean?"
Scale: Number of Literature Citations to a defined [Person's Work] during a defined [Time Span].
Goal [Person's Work = The Academic, Time Span = Each Decade]: Over 1,000 <- Prof. H G. "That is glory!"

Synonyms: Plan *109: Historic usage only; Planned Level *109: Historic usage only; Goal Level *109: See Level *337; Planned Goal *109.
Related Concepts: Aim *001; Target *048; Stretch *404; Wish *244; Ideal *328; Objective *100.
Keyed Icon: > "A single arrowhead, on a performance arrow, pointing towards the future. It is the same icon as for Budget *480 (which is on a resource arrow, --->--->O).
In context: O---->---->
Always use an output arrow from a function oval to represent a performance attribute. The Goal icon is the '>' on the scalar arrow. If other scalar levels are shown, the positioning of the tip of the icon symbol should reflect the Goal level relative to these other levels."

Icon **Concept *161**

In Planguage, an icon is a symbol, that is keyed (Keyed Icon) or drawn (a Drawn Icon), that represents a concept. All icons are graphic or pictorial in nature – they should not use words or national languages.
Related Concepts: Keyed Icon *144; Drawn Icon *085; Symbol *161.

IE **Concept *283**

Acronym for 'Impact Estimation'.
Historical Note: History of the development of Impact Estimation: I've developed the IE method in the course of my consultancy work. I originally started in the early 1960s with multidimensional evaluation models, which were later published (in 1968 at the Nord Data Conference) as 'Weighted Ranking by Levels' or the MECCA method. By the 1970s, I had adopted a table format. This was chiefly inspired by the 'Requirements/Properties Matrix' presented by Dr. Barry Boehm, then of TRW Systems, in his 1974 IFIP Speech in Stockholm. The main difference in my approach was that I wanted to provide a more quantified method; while I liked the idea of the matrix structure, I found the TRW implementation too fuzzy. See my 'Software Metrics' book (Gilb 1976 Out of Print). Finally, by the mid-1980s, I had the basics of Impact Estimation (IE), which was described in my Principles of Software Engineering Management (Gilb 1988) book. Subsequently, during the early 1990s, we (Kai Gilb and I) added credibility evaluation and the use of graphical 'skyscraper' representation. I have also recently started using IE to explicitly outline evolutionary step sequences.
Notes:
Key Differences: Impact Estimation and QFD. I am frequently asked to compare IE with Quality Function Deployment (QFD) (Akao 1990). The key difference between IE and QFD tables lies in *the degree of quantification.* In QFD, the objectives are rarely stated quantitatively, design ideas tend not to be formally specified, cost is hardly ever considered and the evaluation of the impact of the design ideas is not numeric (usually only an assignment of 'weak', 'medium' and 'strong' is made). Also, there is no attempt at citing evidence, sources or credibility.

If **Concept *399**

'If' is a logical operator used in qualifiers to explicitly specify conditions.

Notes:

1. The 'If' is *implied* for all terms in a qualifier. However, 'If' may be used to communicate a condition more explicitly to the novice reader.

   Example:
   Goal [USA, If Law 153 Passed]: 99.9%.
   Goal [If Europe, If Product XYZ Announced]: 60%.

Synonyms: IF *399.
Related Concepts: Qualifier *124; Condition *024.

Impact **Concept *087**

An 'impact' is the estimated or actual numeric effect of a design idea (or set of design ideas or Evo step) on a requirement attribute under given conditions.

Notes:

1. Full impact information includes the following: a scale impact, a percentage impact and uncertainty data (known error margins). The additional related information required to support an impact includes the evidence, source(s) and credibility.
2. If an impact is estimated, it is an Impact Estimate *433.

Related Concepts: Impact Estimate *433; Impacts *334.

Impact Estimate **Concept *433**

An impact estimate is an evaluated guess as to the result of implementing a design idea. In other words, it is a considered, quantified guess of the effect on a specific scalar requirement attribute (performance or resource) of implementing a design idea (or set of design ideas) in a system (or system subset) under stated conditions.

A full impact estimate includes the following: a scale impact, a percentage impact and uncertainty data (known error margins). The additional related information required to support an impact estimate includes the evidence, source(s) and credibility.

Notes:

1. An impact estimate can be positive, neutral or negative (undesirable in relation to stated target levels).
2. Note the distinction between a scale impact (an absolute numeric value on a Scale), and a percentage impact (the percentage improvement estimated to be achieved in moving from the chosen baseline towards the chosen target).
3. An impact estimate is usually concerned with the system improvement, rather than with stakeholder value (however, this depends on the choice of requirement attribute: stakeholder value can be tackled, for example, Financial Saving).
4. An impact estimate is usually on a scalar requirement. However, much more rarely, it can be on a binary requirement of the system (that is, on a function requirement, a design constraint or a condition constraint). This is used in situations where an explicit check is considered necessary to help evaluate design ideas.

Abbreviations: Impact *433 "Often 'Impact' is short for 'Impact Estimate'. See also Impact *087."
Related Concepts: Impact *087; Scale Impact *403; Percentage Impact *306; Side Effect *273.

Impact Estimation **Concept *283**

A Planguage method/process used to evaluate the quantitative impacts of design ideas on requirements.

Description: Chapter 9, "Impact Estimation: How to Understand Strategies and Design Ideas."

Acronym: IE *283.

Impacts **Concept *334**

The 'Impacts' parameter is used to identify the set of attributes that are considered likely to be impacted by a given attribute (usually another requirement attribute or a proposed design idea).

Notes:

1. 'Impacts' can be used to capture Impact Estimation table relationships before actual numeric estimation.
2. 'Impacts' differs from 'Supports' in that it can be used to identify *side effects, including negative side effects*, as well as the intended direct positive impacts.

Example:
Design Idea 1: Handbook Impacts {Learning, Development Cost}.
or
Design Idea 1: Handbook -> {Learning, Development Cost}.

Keyed Icon: ->

"The 'Impacts' arrow is only valid in the context of tags referring to things that can impact one another."

Example:
Design A -> Requirement B.

Related Concepts: Supports *415; Is Impacted By *412.

Includes **Concept *391**

'Includes' expresses the concept of inclusion of a set of components within a larger set of components. 'A Includes B.' means that B is a subcomponent of the component A.

Example:
B: Includes {C, D, E}.

Example:
Bee.Wings "Wings is a member of supra concept, or parent concept, Bee."
Bee: Includes {Wings, Legs, Eyes, Sting, Body}.
Bee: {Wings, Legs, Eyes, Sting, Body}. "Includes is implied by the set parenthesis."
Alternative formats and synonyms for Includes.

Related Concepts: Consists Of *616.

Keyed Icon: { } "In context, X: {A, B} means X includes A and B."

Incremental Development **Concept *318**

Incremental development means designing a system largely up-front, and then dividing its construction, and perhaps handover, into a series of cumulative increments.

Notes:

1. Incremental Development is defined here in order to contrast it with, and distinguish it from, Evo:
   - Incremental development differs from Evo in that most all of the Incremental Development requirements design effort is up-front. In contrast Evo carries out requirements and design detailing *gradually* in each Evo cycle
   - Incremental development is without Evo's intent of *measuring* the progress of each (incremental) step fully (for example, measuring delivered performance levels), then *learning* from these feedback measures and, *changing* the requirements and/or design accordingly
   - Incremental development is also without the intent of delivering the steps (increments) with the highest 'value to cost ratio' or 'performance to cost ratio' first.
2. It is unfortunately common practice to say or write 'incremental' when the strictly correct term according to the distinctions defined here is 'evolutionary.' Indeed, all evolutionary processes are also incremental, but they are a subclass deserving distinctive terminology to announce the differences. This 'lazy' use of the term is a sure sign of people who do not have deep understanding, or concern for, the value of feedback and change. Beware of their advice or opinions! The US DoD (DoD Evo 2002 http://www.acq.osd.mil/dpap/ar/1_multipart_xF8FF_2_EA%20SD%20Definitions%20final.pdf), among others, has taken the trouble to carefully distinguish these concepts!

Related Concepts: Evolutionary Project Management (Evo) *355.

Incremental Scale Impact — Concept *307

For a scalar requirement, this is the numeric impact of a design idea *relative* to the specified baseline level. If there is a negative impact, then the numeric value will be negative.

Scale Impact – Baseline = Incremental Scale Impact

Example:

Consider an objective concerning say, a 'Customer Response Time,' with a defined Scale of 'Minutes to Wait.' If the Baseline was 'Past: 20 minutes to wait' and the Target was 'Goal: 5 minutes to wait' and, the Scale Impact (estimated or actual) of Design Idea X on Customer Response Time was a result of '12 minutes to wait,' then the Incremental Scale Impact is 8 minutes (20 − 12 = 8).

The Percentage Impact is 8/15 or 53% relative to the Baseline (0%, or 20 minutes) and to the Target (100%, or 5 minutes).

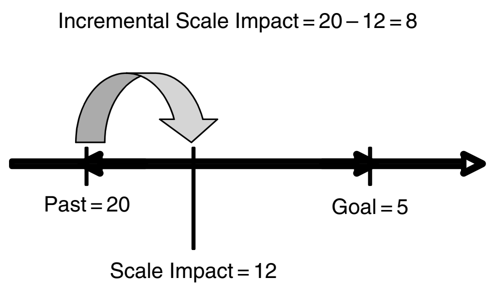


Figure G10

Notes:

1. Designs vary in their impact, depending on previous circumstances. The incremental impact is a function of these circumstances. The impact of a design idea is *not a constant*, irrespective of the circumstances it is implemented in.

Related Concepts: Percentage Impact *306; Scale Impact *403.

Inspection — Concept *051

'Inspection' is a synonym for Specification Quality Control (SQC).

Notes:

1. Michael Fagan originated the term 'Inspection' in connection with software within IBM. He developed the initial method for quality control of software. It is based on the work of Walter Shewhart, Joseph Juran and others, who used the term for quality control of products (rather than of specifications). Given the confusion in engineering environments over the use of the term 'Inspection' (to hardware engineers it means quality control after production of something), I prefer to use the term, SQC.
2. Many people incorrectly equate the Defect Detection Process (DDP) with 'Inspection.' They omit the Defect Prevention Process (DPP). This is because they are unaware of the additional developments to Inspection introduced by Mays and Jones (Mays 1995).

Synonyms: Specification Quality Control (SQC) *051; Peer Review *051.

Is Impacted By — Concept *412

'Is Impacted By' is used to indicate any *other* specified items (such as requirements, objectives, designs, policies or conditions), which affect, or might affect, a defined specification itself, or what it refers to.

Notes:

1. The purpose of 'Is Impacted By' is to help in risk identification and analysis. We are trying to explicitly identify and document factors, which we believe influence the results. This will hopefully result in specific action or design to keep those impacts from threatening our planned results.
2. The more general purpose of Is 'Impacted By,' and many other Planguage relationship mechanisms, is to build a 'web of connections' between specifications (that is, between system components). This web of connections serves many purposes. Risk management was mentioned above. Other uses are configuration management, system familiarization, quality control, estimation, contracting, prioritization, and reviewing.

   Example:
   A:
   Is Impacted By: {Help Desk Capacity, User Motivation, User Training, Bug Frequency}.
3. 'Is Impacted By' is differs from considerations of Risk/Threat in that both *good* and bad impacts are considered. With Risk/Threat, we are primarily concerned with the potential for *negative* impacts.

4. For strong primary intended impacts, the 'Is Supported By' icon can be used, A ->> B. meaning B is supported by A. In other words, A is *primarily* the way we intend to achieve the requirement/value B.

Synonyms: Impacted By *412.

Related Concepts: Risk *309; Threat *309; Dependency *189; Impacts *334; Supports *415; Is Supported By *414.

Is Part Of **Concept *621**

'Is Part Of' is a parameter, which indicates that a specification is a component or element of some other component or element.

Example:
PDSA Cycle:
Type: Process.
Consists Of: Sub-Process {Plan, Do, Study, Act}.
Plan:
Type: Sub-Process.
Is Part Of: PDSA Cycle.

Example:
Reliability Is Part Of Availability.

Related Concepts: Consists Of *616; Includes *391; Component *022; Element *022.

Is Supported By **Concept *414**

'Is Supported By' is used to list the tags of any and all attributes that contribute usefully to the accomplishment of the planned target levels of a defined requirement.

Notes:

1. The attributes that can provide support include designs and Evo steps.

   Example:
   Goal X:
   Scale: <some scale definition>
   Goal [Next Version]: 55%.
   Is Supported By: Design Idea {A, B, C}.

Synonyms: Supported By *414.

Related Concepts: Supports *415; Impacts *334; Is Impacted By *412.

Issue **Concept *276**

An issue is any subject of concern that needs to be noted for analysis and resolution.

Example:
ISS1: Issue: We have not analyzed risks and dependencies yet.

Notes:

1. A specification issue is an element of written specification, which we suspect violates a specification rule. It is noted for later resolution. It will be resolved by being declared to be either a defect (a rule violation) or as requiring no further action.

Related Concepts: Resolution *525; Specification Issue *529.

Kin **Concept *353**

Kin specifications are specifications, which derive from an identical set of source specifications.

Notes:

1. For example, test plans, source code and user handbooks could all be derived from the same requirements or the same design.
2. Kin specifications can serve as additional information to perform defect checking in the Specification Quality Control (SQC) process. For example, United Defense in Minnesota reported [personal communication] that their software engineering checked the program code against their test cases, both derived from the same requirements. They reported that they usefully found major defects in both these kin documents.

Landing Zone **Concept *605**

A landing zone is a target range that stretches from just better than a Fail level through the Goal/Budget level to the Stretch Level.

Notes:

1. A landing zone is analogous to a parachute's landing zone. A range that we realistically hope we can land in somewhere. This avoids the simplified notion of an exact Goal/Budget being the target.
2. For a set of requirements, the overall landing zone is the set of landing zones, which 'creates a space' over all the requirement dimensions.
3. The multidimensionality of landing zones is an important feature. The space below Goal may seem unacceptable, but when you consider all dimensions at once, sub-par achievement in a single dimension is completely acceptable, if it means optimal system performance.
4. A landing zone covers a success range and an acceptable range.

Related Concepts: Range *552.

Historical Note: The source of the Landing Zone concept was Intel, Oregon (via Erik Simmons, 2002).

Table G2 A simple example showing multidimensional landing zones. It is landing within all the landing zones simultaneously that is the aim. A teaching example using fictional data. (Courtesy of Erik Simmons, Intel).

Attribute	*Fail*	*Goal*	*Stretch*
Price	>$27000	$20000	$17,500
Mileage (City)	<18 mpg	25 mpg	35 mpg
Seating	<4 adults	5 adults	6 adults
Interior Noise at 65 mph	>74 dBA	65 dBA	55 dBA
Projected 3-year Maintenance Cost	>$3000	$2000	$1500

Level **Concept *337**

A level is a defined numeric position on a scale of measure.

Notes:

1. A scalar level applies to either a performance or a resource attribute.
2. A level on a scale of measure indicates one of the following:
   - a benchmark: an actual measurement or estimated level in the past
   - a target: a requirement level
   - a constraint: a limit
   - an estimate of the impact of a design idea

Synonyms: Point *337: A position on a Scale.

Related Concepts: Range *552; Goal *109: Goal Level; Budget *480: Budget Level; Stretch *404: Stretch Level; Wish *244: Wish Level; Fail *098: Fail Level; Survival *440: Survival Level; Catastrophe *602: Catastrophe Level; Past *106: Past Level; Record *127: Record Level; Trend *155: Trend Level; Limit *606: An extreme boundary of the range of a level.

Keyed Icon: | "In context on a Scale: ----|---> This is the generic attribute level icon. It can be used instead of any of the more specific level icons (for example, '>' for Goal or Budget)."

Limit **Concept *606**

A limit is a numeric level at a border, that is, at an edge of a scalar range (a success range, an acceptable range, a failure range or a catastrophe range). It is specifically used at the edges of ranges associated with constraints: fail limit and survival limit/ catastrophe limit.

Related Concepts: Range *552; Fail *098: Fail Limit; Survival *440: Survival Limit; Catastrophe *602: Catastrophe Limit.

Logical Page **Concept *103**

A logical page is defined as a defined number of *non-commentary* words. Default Volume: If no other definition is given, use '300 non-commentary words' per logical page as default.

Rationale: This measure of specification 'volume' is used to make sure that varying page sizes and page content does not cause false volume measures. Volume measures are important for establishing checking rates (logical pages per hour) and defect density (majors per logical page).

Notes:

1. 'Non-commentary' is a useful concept because it only pays off to worry about optimum checking rates or defect densities on non-commentary specification (where potential danger lies in defects).

Abbreviations: Page *103: In an SQC context.

Acronym: LP *103.

Synonyms: Logical Page Size *103.

Related Concepts: Physical Page: This is one side, facing a reader, of space for textual and/or graphical symbols, of physical or electronic nature. It has clearly defined borders, traditionally rectangular, with any arbitrary quantity of symbols.

Major Defect **Concept *091**

A major defect is a specification defect (a rule violation), which if not fixed at an early stage of specification, then it's consequences will possibly grow substantially, in cost-to-fix and/or damage potential. A

major defect has on average approximately an order of magnitude more downstream cost potential than it's cost to remove immediately.
Rationale: This concept and classification is necessary to help SQC checkers and other QC people to focus on what defects it pays off to find and eliminate in a specification. Without this classification, up to 90% of QC effort might be wasted dealing with minor defects.
Abbreviations: Major *091; M *091: Often intentionally written with a capital 'M'.
Related Concepts: Specification Defect *043; Minor Defect *096.

Master Definition — Concept *303

The master definition of a specification or specification element is the primary and authoritative source of information about its meaning. The master definition overrides any other (informal, not master) definition that is in conflict with it.

Notes:
1. This glossary contains master definitions, but the definitions themselves may contain explanations of other terms (for example, in the Related Concepts sections), which are less formal and less authoritative than the master definition for that concept.
2. A master definition should contain full information about the source, authority, version and status, where relevant.
3. It is good practice to only permit a single master definition for a term to exist, and all references concerning the master definition must point to that single definition.

Example:
Master Definition: The primary correct source of a term's meaning.
Type: Master Definition.

Measure, To — Concept *386

To measure is to determine the numeric level of a scalar attribute under specified conditions, using a defined process, by means of examining a real system.

Notes:
1. Measurement is done on the defined Scale, with respect to specified qualifier conditions.
2. Measuring is done using defined Meters.

Example:
Usability:
Scale: Mean Time To Learn.
Meter [Experts]: Use the upper 5% of our experienced staff in tests.
Meter [Novices]: Use 10% of current year's intake of new people.
Fail [Experts, Complex Task]: 15 minutes.
Goal [Novices, Simple Tasks]: 10 minutes.
Two different Meter specifications are made in order to make it clear how the two different targets shall be measured.

3. Measuring is distinct from quantification and estimation. Quantification is merely defining an attribute with the help of a scale of measure, and benchmarks and/or target values. Estimation is trying to determine results based on past data.

Related Concepts: Scale *132; Meter *093; Quantify, To *385; Estimate, To *059.

Meter **Concept *093**

A Meter parameter is used to identify, or specify, the definition of a practical measuring device, process, or test that has been selected for use in measuring a numeric value (level) on a defined Scale.

"there is nothing more important for the transaction of business than use of operational definitions."
(W. Edwards Deming, Out of the Crisis *(Deming 1986))*

Example:
Satisfaction:
Scale: Percentage of <satisfied> Customers.
Meter [New Product, After Launch]: On-site survey after 30 days use for all Customers.
Past [This Year, USA]: 30%.
Meter [Past]: Sample of 306 out of 1,000+ Customers.
Record [Last Year, Europe]: 44%.
Meter [Record]: 100% of Customers.
Goal [After Launch]: 99% <- Marketing Director.
In the above example, the first Meter specification is the one that will be assumed, in default of any other specification, particularly for use in validating the achievement of Goal targets. Both the benchmarks (Past and Record) have local Meter specifications, which tell us more exactly the measuring process used to gather their data. Of course, this implies that these benchmark and target numbers are not as comparable as we would like them to be. But that is the way it often is, and our local Meter specifications at least allows us to judge whether this difference is significant for our current purposes.

Keyed Icon: -|?| – "A '?' on top of a Scale icon, -|-|-."

Metric **Concept *095**

A metric is any kind of *numerically* expressed system attribute. A metric is defined in terms of a specified scale of measure, and usually one or more numeric points on that scale. The numeric points can be expressed with defined terms that can be translated into numbers. For example, 'Record +10%.'

Normally there will also be other parameters and qualifiers, which add background detail to the metric. For example, Meter and Assumption.

A metric specification encompasses *all related elements* of specification, not just the Scale of the numeric attribute.

A complex specification, with a set of scales of measure, is also a metric expression. There is no implication that it is elementary (has only a single Scale).

Notes:

1. Metrics are used to express 'concepts of variability' clearly – in particular more clearly than mere words (Gilb 1976).

   Example: [Metric Expressions]:
   Scale: Mean Time Between Failures.
   Use: Scale: Time to Learn defined [Average Task]. Past: 30 minutes.
   Design A: Impacts Requirement B: 30%.
   Each of these 3 statements is based on a 'metrics culture.'

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

Example: [Non-Metric Expressions]:
"Reliability"
"Easy to learn"
"Very effective design"
Each of these 3 expressions is based on a 'non-metrics culture.' Nice words – no numbers expressed, defined or implied.

2. The rationale for using metrics includes:
   - to increase *clarity and unambiguousness* of specifications
   - to increase *sensitivity* to small changes in specifications and the system itself
   - to enable systems engineering *logical thinking* about relationships – for example the relation of designs to requirements
   - to provide a better basis for *legal contracting* about systems
   - to enable *evolutionary tracking* of progress towards goals
   - to enable a *process of learning* within projects, engineering and management domains
   - to force engineers to *think* more clearly and *communicate* more clearly with others
3. A metric can be used to express the numeric impact of a *design* on a performance requirement (for example, when using the Impact Estimation method).

Related Concepts: Scale *132; Meter *094.

Minor Defect — Concept *096

A minor defect is a non-major defect. It has no major downstream cost potential.

Notes:
1. A defect which, if not removed at a given time, can be removed later (for example, in test phases or in customer use) at approximately the same cost or penalty.
2. There is little value in dealing with it immediately after it occurs. It can be left to chance, or ignored until it surfaces of its own accord.
3. Minor does not refer to the size of the defect, but to the potential consequences of it downstream.

Abbreviations: Minor *096; m *096: Deliberately written with a small 'm'.
Related Concepts: Major Defect *091.

Mission — Concept *097

A mission specifies who we are (or what we do) in relation to the rest of the world. It is the highest level of function of a system. The mission should not contain 'vision' description ('We make the best planes in the world'). It is an undramatic statement of the main function of a business or organization.
Mission, like function, intentionally excludes specific levels of attributes in its description.

Example:
Mission: We make semiconductors.
Mission: We provide business solutions in manufacturing software.

Related Concepts: Function *069.

Non-Commentary **Concept *294**

'Non-commentary' refers to written specification that is not commentary: it is either core specification or background specification. All major specification defects are found in non-commentary. But, not all specification defects in non-commentary specifications are major. By definition, major defects *cannot* be found in 'commentary.'

Notes:

1. Text diagrams or symbols that are secondary to the main specification purpose, and which do not lead to 'real product' are 'commentary.'
2. Non-commentary sections of a specification can be termed 'meat,' and commentary sections can be termed 'fat'. Commentary includes {notes (like footnotes), comments ("like this"), remarks (Note), introduction, and references (Source)}.
3. Checkers, in SQC, should concentrate on carrying out rigorous checking, at optimum checking rates, on the non-commentary territory. This gives better efficiency in finding defects.
4. It is important to formally distinguish between non-commentary and *commentary*. Authors/writers need to try to make the distinction visually for readers (for example, by using plain text for non-commentary and *italics for commentary*). When the visual distinction is made, and it is clear what is commentary and what is not, then quality control analysts can more easily, and more certainly, decide which defects are major and which are not. They can more quickly scan the commentary and more carefully study and cross-reference check the core specification, and then, somewhat faster, the background specification.

Related Concepts: Commentary *632; Background *507; Core Specification *633; Specification *137; Major Defect *091.

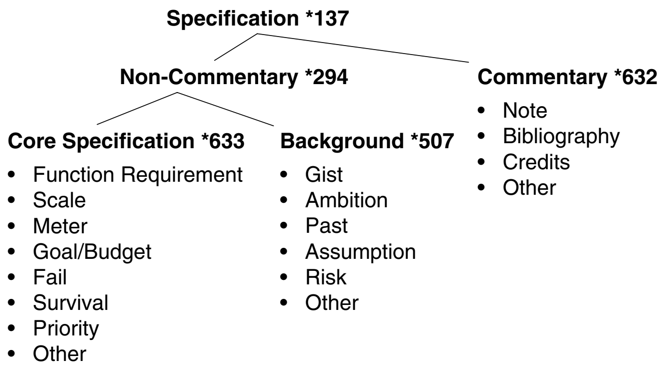


Figure G11
The diagram shows the relationship amongst the different categories of specification.

Note **Concept *018**

A 'note' is a comment or any text that makes any kind of remark related to any statement.

Ways of specifying notes include: *italics*, the use of ''quotation marks'', and the Note parameter (which has a synonym of 'Comment').

Example:
Ambition: Main share of the market. "This is just an example of a comment using quotes in a background statement."
Note: This is an example of the use of the Note parameter.

Notes:

1. Notes must be distinguished from the 'significant' core specification (for example, Goal and Scale) and from 'background' specification (for example, Source, Evidence and Gist). The main reason for this being that a defect in Note specification is usually only a minor defect. Any SQC checking should concentrate on the specification that is *not* Note specification (that is, non-commentary–core and then background), as that is where the major specification defects will be found.

   Example:
   Goal [First Release]: 60% <- Marketing Director [June 6 200X]. "*Source is background, but good for credibility and SQC.*"
   Source: The Encyclopedia.

Synonyms: Notes *018; Comment *018; Remark *018.
Related Concepts: Commentary *632.
Keyed Icon: "…" "Double quote marks around the note."

Objective **Concept *100**

Objective is a synonym of Performance Requirement. See Performance Requirement *100.

Or **Concept *514**

'Or' is a logical operator used in qualifiers, or other appropriate specifications, to indicate alternative conditions. If any one 'Or' condition is true, then the set of conditions is true.

Example:
Stretch [If Multinational Or Government]: 99%.
Stretch [If Multinational or Government]: 99%.
Stretch [If Multinational OR Government]: 99%.

To make a statement read better, the lead capital letter may be dropped, giving 'or.' It can also be spelled all capitals, 'OR,' to emphasize that it is a Planguage logical operator and not a simple text word.

Notes:

1. Parenthesis: {Set parenthesis} and (ordinary mathematical parenthesis), may be used to limit and clarify the extent of a logical expression.

Or Better **Concept *550**

'Or Better' is an expression used within a scalar specification to explicitly emphasize that the specified level has a range of acceptable values, rather than being just a fixed, single value. In other words, 'Or Better' helps identify the specified level as the beginning of a desired range.

'Or Better' is actually implied by a scalar target specification, but it can be useful to be more explicit.

Example:
Goal [Mechanical]: 60 degrees Or Better.
Stretch: 99.90% Or Better.
Survival [Offices]: 35 degrees C Or Better.

Related Concepts: Until *551; Or Worse *549; Or Better *550; Range *552.

Or Worse **Concept *549**

'Or Worse' is an expression used within a scalar specification to explicitly emphasize that the specified level has a range of unacceptable values, rather than being just a fixed, single value. In other words, 'Or Worse' helps identify the beginning of a 'no go' range. 'Or Worse' is actually implied by a constraint specification, but it can be useful to be more explicit.

Example:
Must Avoid [EU, Next Generation Product]: 50% Or Worse.
Fail [Banking Market]: 20% Or Worse.

Related Concepts: Until *551; Or Better *550; Range *552.

Owner **Concept *102**

An owner is a person or group responsible for an object, and for authorizing any change to it.

The parameter, Owner, can be used to explicitly identify ownership.

Notes:

1. For example: a system owner, a specification owner, a standard owner or a process owner.
2. An owner is responsible for updating or changing an object, including maintaining its control information (for example, Status, Version, Quality Level and Location).
3. An owner will ensure the object adheres to any relevant standards.

Related Concepts: Stakeholder *233.

Parameter **Concept *105**

A parameter is a Planguage-defined term. Parameters are always written with at least a leading capital letter, to signal the existence of a formal definition.

Notes:

1. The master definition of most of the Planguage parameters is found in this Glossary. See also http://www.gilb.com for additional, updated and new parameters.
2. A project or specification author can declare and define tailored parameters (as part of a Project Language – a specific project specification language). These can then be reused anywhere in a specification where they are understood. They may in time be officially adopted by some local dialect of Planguage.
3. Parameters are not user-defined terms. User-defined terms are defined by a project or organization, to describe the target system, organization or project. User-defined terms are not part of the definition of a specification language.

Synonyms: Specification Language Parameter *105.

Related Concepts: Term *151; Project Language *247; User-Defined Term *530.

Past **Concept *106**

A Past parameter is used to specify historical experience, a 'benchmark'.
A Past specification states a historical numeric level, on a defined Scale, under specified conditions [time, place, event] for a scalar attribute.

Notes:

1. Past values are stated to give us some interesting benchmark levels for our old system(s) and our competitors' systems.
2. Even 'current' values should be expressed using Past, because immediately they are stated, they are 'past' values. Qualifiers will make plain the currency of a specification.

Rationale: If we did not take the trouble to analyze and specify the past values then we might not set reasonable targets. Unintentionally, targets might even be specified worse than they were in the Past.

Synonyms: Past Level *106.

Related Concepts: Benchmark *007.

Keyed Icon: < "A single arrowhead, normally on a scalar arrow (<----<----O---<---->), pointing 'back' to the past. Note the '<' alone in other contexts has other meanings such as: '<' less than, '<-' (source arrow), '<--------' (tip of scalar arrow). So either it must be used in an unambiguous context or manner, or there be at least one hyphen, or [qualifier], on either side of the arrowhead to distinguish this icon."

PDSA **Concept *168**

Acronym for Plan-Do-Study-Act Cycle *168.

Percentage Impact **Concept *306**

A percentage impact is an incremental scale impact expressed as a percentage of the required improvement (the required improvement being the scalar distance between a chosen benchmark (0%) and a chosen target (100%)).

A percentage impact is part of an impact estimate and is used in Impact Estimation tables.

Synonyms: Incremental Percentage Impact *306; Percentage Incremental Impact *306; %Impact *306.

Related Concepts: Incremental Scale Impact *307; Scale Impact *403.

Percentage Uncertainty **Concept *383**

Percentage 'Uncertainty' is calculated from the scale uncertainty, baseline and target data; and stated together with the percentage impact value.

Notes:

1. Percentage Uncertainty can be used to identify risks in using specific design ideas: the numeric 'best case' and 'worst case' deviations from the Percentage Impact estimate provides important pointers towards the level of risk involved. This can lead to one or more direct actions, if the risk level is judged too high. For example (actions):
   - to specify a design better
   - to change the design itself
   - to get more information about the design impacts
   - to write contract conditions controlling the impact expected
   - to schedule this design for early evolutionary step delivery (so we can see what it really does).

Example:
If Percentage Impact = 30% and Percentage Uncertainty = ± 40%, the overall impact in percentage terms is usually stated as 30% ± 40%. In other words, Percentage Impact is assessed to vary in practice anywhere between −10% and 70%.
Dual System -> Reliability: 30 ± 40% <- Company Experience ranges from –10% to 70%.

Performance — Concept *434

System performance is an attribute set that describes measurably 'how good' the system is at delivering *effectiveness* to its stakeholders. Each individual performance attribute level has a different 'effectiveness,' for different stakeholders and, consequently, different value or utility.
Within Planguage, performance attributes are scalar and are of three types:

- Quality: 'how well'
- Resource Saving: 'how much saved'
- Workload Capacity: 'how much'

Other possibilities exist for defining performance. For example:

> "Performance. A quantitative measure characterizing a physical or functional attribute relating to the execution of a mission or function. Performance attributes include quantity (how many or how much), quality (how well), coverage (how much area, how far), timeliness (how responsive, how frequent), and readiness (availability, mission readiness). Performance is an attribute for all system people, products, and processes including those for development, production, verification, deployment, operations, support, training, and disposal. Thus, supportability parameters, manufacturing process variability, reliability, and so forth, are all performance measures."
> *Source: USA MIL-STD 499B Draft 1992.*

Notes:

1. The system engineer or system stakeholder can select, define, invent, tailor or develop any number of useful or interesting performance measures, to serve the purposes of their current task, or systems engineering process.

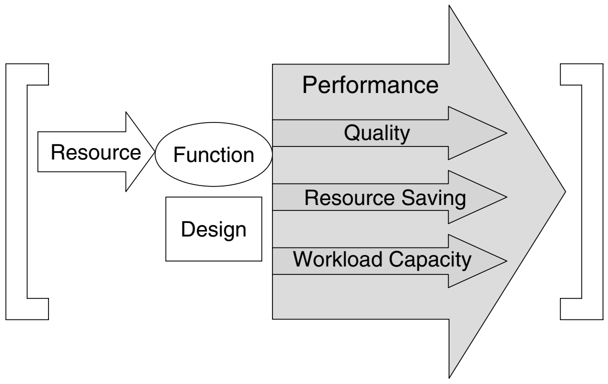


Figure G12
Performance characteristics are classified into three major types within Planguage. This is an arbitrary, but useful distinction. See also the diagram in Quality *125.

2. Performance is intended to cover absolutely all performance measures. It is not limited to the narrower conventional set of performance measures (for example, throughput speed), but explicitly includes the *qualitative* measures of performance, which are so weakly represented and too rarely quantified in conventional thinking.
3. Performance is the most general sense of how well a function is done. Performance includes:
   - quality characteristics (such as availability, usability, integrity, adaptability and portability), and
   - resource saving characteristics (such as cost reduction and reduced elapse times), and
   - work-capacity characteristics (such as storage capacity, maximum number of registered users and transaction execution speed).

Related Concepts: Quality *125; Resource Saving *429; Workload Capacity *459; Performance Requirement (Objective) *100; Performance Target (goal) *439; Performance Constraint *438; Benchmark *007; Target *489; Constraint *218.

Keyed Icon: O---> or O+ "Compare with Keyed Icon [Resource *199]: --->O or –O and Keyed Icon [System *145 [Not Design]]: {Resource, Function, Performance, Condition}: [--->O--->] or [–O+]."

Performance Constraint — Concept *438

A performance constraint specifies some upper and lower limits for an elementary scalar performance attribute. These limits are either levels at which failure of some kind will be experienced, or levels at which the survival of the entire system is threatened.

Fail and Survival parameters are used to specify performance constraints.

Notes:

1. Stakeholders impose constraints. These stakeholders and their motivation should be explicitly documented together with the constraint specification (for example using Authority, Source, Rationale or Stakeholder parameters).

   Example:
   Speed:
   Scale: Time in seconds <to react>.
   Survival [Public Places]: 10 seconds maximum.
   Authority: Public Safety Law.
   Fail [All Uses of our Product]: 5 seconds.
   Authority: Our Quality Director. Rationale: Time to react to alarm light.
   Goal [Public Places]: 4 seconds. <- Project Manager.
2. Performance *constraint* (Survival and Fail) levels usually lie 'outside' the performance target (Goal, Stretch, Wish) levels.
3. Why an upper constraint limit for a performance? There are many reasons, which include:
   - To avoid 'arms race escalation'
   - To avoid unnecessary costs, which would not be valued by a market
   - To avoid costing yourself out of a market
   - To avoid unnecessary cost – contract income is constant when you reach such a limit
   - To avoid 'gold plating' and over-engineering
   - To avoid becoming a monopoly and provoking legal reaction
   - To avoid showing your hand to the opposition.

Related Concepts: Performance *434; Performance Requirement (Objective) *100; Performance Target (goal) *439; Constraint *218; Fail *098; Survival *440.

Performance Requirement **Concept *100**

A performance requirement (objective) specifies the stakeholder requirements for 'how well' a system should perform.

A performance requirement can be complex or elementary. It is a scalar concept, and at elementary level is defined quantitatively by a set of performance targets and performance constraints.

Typical examples of performance requirements include 'Usability,' 'Reliability' and 'Customer Satisfaction.'

Performance Requirements are limited to consideration of the *performance* effectiveness of a system, without regard to the efficiency of it. That is, a performance requirement describes some aspect of the required performance; it does not describe the costs (the resources needed) to get the performance.

Notes:

1. The distinction between use of the terms 'objectives,' 'quality requirements' or 'performance requirements' is often simply dependent on the culture using them. Engineers are more likely to speak about 'quality requirements' for a system or product. Managers (of people and organizations) are more likely to think in terms of business/technical 'objectives.'
2. A performance requirement is a potentially complex, detailed specification. It can consist of a whole hierarchy of performance attributes.
3. For each *elementary* performance attribute (distinguished by having only one Scale), there can be *many* performance targets and/or performance constraints.

   (Note: this does not mean that the concept behind an elementary performance requirement is not 'really' somehow complex, and that

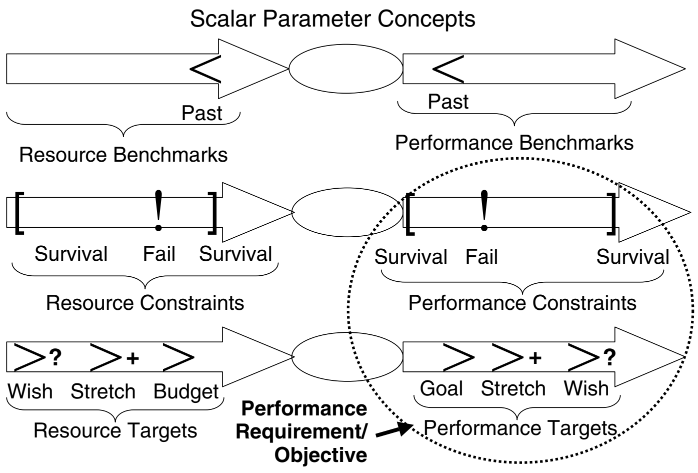


Figure G13
A performance requirement (objective) is specified as a set of performance targets and performance constraints.

it is not *capable* of further sub-setting. It is simply that for the given system, further sub-concepts are not considered to be of interest or use at the current time. So 'complex' means 'complex in terms of whether we have *decided to* decompose the concept in the current specification,' not complex in terms of some constant reality.)

Example:
We can generally speak about a performance requirement, 'Reliability' for a defined system. Reliability may well be specified as elementary (having only one scale of measure). There can be several targets and constraints specified. Here below, is an elementary Reliability performance requirement with a Fail level and two Goal level specifications. Qualifiers distinguish the Goal level specifications from each other.
Reliability: "A Performance Requirement or Objective"
Scale: MTBF.
Fail: 30,000 Hours. "*Constraint 1*"
Goal [1st Release]: 40,000 Hours. "*Goal 1*"
Goal [2nd Release]: 50,000 Hours. "*Goal 2*"
Of course, the 'Reliability' performance requirement could instead be a complex objective (that is, composed of one or more sub-objectives (elementary or complex)). For example, we might be interested in two Scales: 'Mean Time Between Failure (MTBF)' and 'Number of Repeat Occurrences of Faults.' We would specify both of them as sub-requirements of the 'Reliability' requirement.

Example:
Reliability: {Failure Rate, Repeat Failures}.

4. Additional supporting information can be present in benchmark parameters (Past, Record, Trend).

Example:
Stretch [Main Markets, Within the Decade]: 99.998%.
A Performance Target.

Example:
Security [Corporate Webservers]: Elementary Performance Requirement.
Version: 3.1. Owner: Tom. Sponsor: Simon.
Scale: Annual Frequency of defined [Type of Penetration] using defined [Type of Threat] used by defined [Type of Perpetrator].
Meter [Acceptance]: At least 300 representative cases of [Type of Threat] <- Contract 2.3.5.
================== Performance Targets ==================
Goal [Security Type 1, Next Year]: <10 <- Official Project Steering Committee Agreement.
Stretch [Security Type 1, Next Year]: 0 <- Technical Director's Challenge.
================= Performance Constraint ================
Survival [Security Type 1, Next Year and On]: 60 <- CEO Public Promise of Improvement.
====================== Benchmark =====================
Past [Security Type 1, Last Year]: 66 <- Annual Executive Security Report [Page 55].
====================== Definitions =====================
Security Type 1: Defined As: Type of Penetration = Access, Type of Threat = Remote Terminal, Type of Perpetrator = Hacker].
An Elementary Performance Requirement.

> *Performance Requirement.* "The extent to which a mission or function must be executed, generally measured in terms of quantity, quality, coverage, timeliness, or readiness."
>
> *Source: US MIL-STD 499B 1994.*

Synonyms: Objective *100.
Related Concepts: Requirement *026; Performance Target (goal) *439; Performance Constraint *438.

Performance Target — Concept *439

A performance target (goal) is a stakeholder-valued, numeric level of system performance. There are three types of performance target:

- a committed planned level (Goal),
- an uncommitted motivating level (Stretch), and
- an uncommitted valued level (Wish).

Notes:

1. The target parameters {Goal, Stretch, Wish} are used to express performance targets.
   Example:
   Goal [Main Asian Markets, Next Quarter]: 60,000 hours.
   A Performance Target.
2. A performance target is a *single* required level of performance (such as a Goal specification).
   Example:
   Goal [USA, Next Release]: 99.50%.
3. In contrast, a performance *requirement* includes both performance *targets* and performance *constraints.*

Synonyms: goal *439 (with a small 'g' to distinguish it from the parameter, 'Goal').
Related Concepts: Performance *434; Performance Requirement (Objective) *100; Performance Constraint *438; Target *048; Goal *109; Stretch *404; Wish *244.

Performance to Cost Ratio — Concept *010

For a design idea (or set of design ideas or an Evo step), the ratio of the performance improvements to the cost of the resources needed to implement them.
For a selected set of requirements:
Performance to Cost Ratio = Sum of Performance/Sum of Costs
Rationale: Such ratios allow comparison of different design ideas to determine which is the most cost efficient. Popularly (USA), "Bang for the buck".
Notes:

1. Provided Sum of Costs is used, costs are any idea of resources, and are not limited to financial costs. This is possible because it is the *percentage* costs that are being summed. Any useful set of cost attributes may be used.
2. An alternative way to calculate the Performance to Cost Ratio is to use the sum of the absolute financial costs, rather than Sum of Costs. This can be a simpler solution as it avoids some arithmetic. It also gives the actual financial costs more prominence.
3. Keep in mind the essential distinctions between achieving the performance requirements, the consequent value to given stakeholders under

given circumstances of reaching those target levels, and the actual benefits that the stakeholders obtain. A 'value to cost' ratio calculation or a 'benefit to cost' ratio calculation, while more demanding, might give a more realistic evaluation.

A performance to cost ratio is a simpler measure: the amount of requirements satisfied to the cost incurred. It could be considered as 'how far a project is going towards meeting all its goals for what level of cost'.

Related Concepts: Sum of Performance *008; Sum of Costs *128; Value *269; Value to Cost Ratio *635; Benefit *009.

Place **Concept *107**

'Place' defines '*where*'. It relates to any notion of place, such as geographic location, stakeholder role, customer market or system component.

Place is used both as a parameter and in a qualifier condition.

Example:

Place [Person Type]: {Buddhist, Korean, Teenager}.

'Place' used as a parameter.

Goal [Place = {USA, CAN, MEX}, Date = Ten Years Time, Software Sub-system]: 60% <- North America Marketing Plan.

'Place' defined as a set of acronyms for countries.

Notes:

1. Place refers to the set of possible scope dimensions that are neither temporal, nor event dependent.
2. Place refers to notions such as:
   - geography (for example, a city, a country)
   - organizational (for example, IBM, Nokia, US Government, UK MoD)
   - types of people, including groups (for example, novice, teenager, pensioners)
   - system components (for example, hardware, radio transmitter, software, database, user terminals, chips)
   - role (manager, operator, trainee).
3. There can be any useful number of place dimensions in a single qualifier expression.

Synonyms: Space *107.

Related Concepts: Time *153; Event *062; Qualifier *124; Scope *419.

Plan-Do-Study-Act Cycle **Concept *168**

The Plan-Do-Study-Act Cycle (PDSA Cycle) is a process cycle. It is a method for changing a defined work process to reflect measurably better process. The concept was developed by Walter Shewhart and taught by W. Edwards Deming (Delavigne and Robertson 1994; Deming 1986; 1993).

Notes:

1. The PDSA Cycle involves the following:
   - Decide on an improvement and how to accomplish it. ('Plan')
   - Carry out the improvement as planned ('Do')
   - Gather data about costs and resulting improvements and side effects, analyze the data and decide what it means ('Study', sometimes called 'Check')
   - Adopt the change as is, or try again in a better way ('Act')

The 'Plan' phase involves specifying a theory or hypothesis, including a procedure for testing it. It sets out the order and timing of *tasks* and *events* that need to occur to achieve the 'Do' phase. In addition, 'Plan' specifies the entry and exit conditions for the 'Do' phase, including the criteria for terminating it early (prior to required results being achieved) and the testing procedures. It also plans the resources required. In Evo, 'Plan' is the planning for the delivery of the step that has just been selected.

The 'Do' phase involves implementing (Do . . . ing) what you have planned, that is, carrying out the 'experiment' to see how your plan measures up to reality. In Evo, 'Do' is the step implementation.

The 'Study' phase observes and gathers data concerning the results of the 'Do' phase. It is basically concerned with 'What happened?' It can involve highly varied analysis activities, depending on the activity being controlled, such as obtaining feedback data, carrying out specification quality control (SQC) and testing. It is a preparation for drawing conclusions and taking action in the 'Act' phase. In Evo, 'Study' is the phase where we analyze all the feedback from the last step, in relation to requirements and external impulses (like change of market or law).

The 'Act' phase decides on what course of action should be taken based on the information supplied by the 'Study' phase. It is to standardize the process at a new level, or to draw new conclusions about our original theory or to determine and select new theories (to design or modify processes). In Evo, 'Act' means reviewing the Evo plan, determining the gap priorities, finding alternative steps and deciding on the next step from the various alternatives. If the results of the last step are not satisfactory, a course of action to correct it might be decided upon.

2. A PDSA Cycle is enabled by standardizing and stabilizing its target process (meaning: 'wherever you carry out the improvement'), so that the effects of any process changes can be credibly observed.

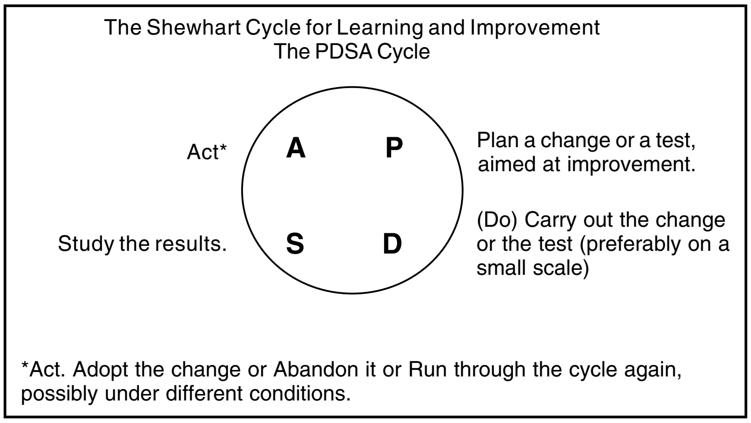

The Shewhart Cycle for Learning and Improvement
The PDSA Cycle
Act*
A
P
Plan a change or a test, aimed at improvement.
Study the results.
S
D
(Do) Carry out the change or the test (preferably on a small scale)
*Act. Adopt the change or Abandon it or Run through the cycle again, possibly under different conditions.

Figure G14
Reproduction from a letter to Tom Gilb from W. Edwards Deming, May 18, 1991.

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

Acronyms: PDSA Cycle *168; PDSA *168.
Related Concepts: Plan [PDSA] *169; Do [PDSA] *170; Study [PDSA] *171; Act [PDSA] *172; Evolutionary Project Management *355; Statistical Process Control (SPC) *466.
Historical Note: The Study phase has sometimes been called 'Check'. For example, Deming used Plan-Do-Check-Act (PDCA) in his book, Out of the Crisis (Deming 1986). However, in a letter to me (Tom Gilb) in May 1991, he said he strongly preferred the initial Walter Shewhart usage, PDSA, due to the word 'check' having other interpretations, such as 'stop.'

Planguage © Tom Gilb — Concept *030

Planguage is a specification language and a set of related methods for systems engineering.
Notes:
1. Planguage specifically supports all aspects of systems engineering including requirement specification, design specification, design impact analysis, specification quality control and evolutionary project management.
2. Planguage can, however, be used much more generally; it has been used successfully to plan such diverse things as family holidays and multinational charity aid project plans.
   Planguage is designed to express ideas about any requirements, designs and plans.
3. Planguage is intended for use throughout a project lifecycle: for planning, problem-solving, specification quality control and result delivery to stakeholders.
4. Planguage has been developed by Tom Gilb, and is defined in this book. There has been lots of feedback from clients and professional friends. Its general content is described more fully in both the Introduction and Chapter 1 of this book.
5. The purpose of the copyright is to avoid being prevented later from using this term by others. Permission to use the term freely is granted by Tom Gilb when © is acknowledged.
6. If any reader finds the term 'Planguage' too 'cute,' they may use the more directly descriptive 'Planning Language.' I (Tom Gilb) often refer to 'Planning Language' before I introduce the term 'Planguage.'

Synonyms: Planning Language *030.

Priority — Concept *112

A 'priority' is the determination of a relative claim on limited resources. Priority is the relative right of a competing requirement to the budgeted resources. If resources were unlimited, there would be no need to prioritize things. You *could* 'have it all.'
An explicit Priority parameter can be used to specify any direct priority relationships.
Notes:
1. The specified, qualified, stakeholder requirements (the targets and constraints stated in the requirements) provide 'natural' (requirement related), and dynamically computable, priority information. The gaps remaining until the goals are met, and budgets used up, can be measured and computed. In general, the largest gap to a performance target will have the highest priority, and the largest gap to using up a budgeted resource will indicate the safest resource opportunity.

However, to determine your priorities in specific cases, the degrees of risk and uncertainties, and/or knowledge of the effort (the resources) needed to close the individual gaps to meet each of the function and performance targets, and the likely resulting stakeholder values on delivery, all need to be taken into account (see below for a more complete list).

2. Priority is not a constant. It cannot and should not be determined and specified in the form of static priority weighting numbers (for example, '25%,' or third on the priority list) or words (for example, 'High'). Current priority depends on how well satisfied the competing performance targets are and how 'used up' the budgeted resources are at any given time. Current priority also depends on the more fundamental changes that can occur in requirements themselves, as stakeholders modify their requirements, and as the external business environment alters – to demand requirement changes.

Example:
Consider your priorities for food and air. If you are hungry, then you give priority to eating. However, as soon as air is in short supply, your priorities change. Your body gives priority to breathing.
Your body knows your food and air requirements, both targets and constraints. Your body knows the current supply levels of these resources. When changes in the body resource levels dictate change in our priorities, this knowledge triggers the body to appropriate action. The body, as a system, acts in order to ensure our comfort and survival. *Priority is dynamic.*

3. Priority is decided by a wide variety of factors, which include but are not limited to:
   - qualifier conditions (factors such as timescales and location)
   - stakeholder authority
   - stakeholder influence
   - consequences of failure (not meeting Fail constraint levels)
   - consequences of catastrophe (not meeting Survival constraint levels)
   - previous experience of meeting similar requirements (including no experience!)
   - complexity of meeting the requirements
   - consequences of success (primarily meeting the *performance* targets, the Goal levels: the system improvements delivered, and the benefits likely to be experienced)
   - resource availability (or maybe more significantly, resource unavailability)
   - dependencies.

   Within Planguage, the relative priority of a requirement depends on a combination of the elements of the specification. These elements include target levels (for example, Goal and Budget), constraint levels (for example, Fail and Survival), its qualifier conditions [time, place, event], and its authority level.

   If you consider only the different requirement level parameters as a class: first priority is satisfaction of all the constraints; the Survival levels, then the Fail levels. Then next priority becomes satisfaction of targets; the Goal levels, after that any Stretch goals are considered, and finally perhaps some Wish levels.

However, you have also to consider the qualifier conditions as well, for all these levels, as qualifiers bring into play additional factors, like the timescales for the requirements to be met, and the events under which the requirements actually exist.

No priority exists until the qualifier conditions [time, place, event] warn us of potentially unfulfilled requirements. Targets and constraints are not finally effective until their qualifier conditions are true, but the designer, the architect and the project manager have to prioritize their contribution in *advance* of deadlines, and other conditions, becoming 'true.'

4. Stakeholders' needs will ultimately decide the relative priorities. There will be trade-offs to consider when there are conflicts between requirements. System designers should evaluate priorities and then present the results for confirmation, selection or conflict resolution to the stakeholders themselves. Stakeholders, as a result of seeing the cost and feasibility of design options, may then choose to change some of their priority specifications.
5. The Priority parameter can be used to help people to more directly understand the priorities (or to confirm the derived priorities with stakeholders). Alternatively, it can be used to specify priorities that differ from what would otherwise be expected or evaluated.
6. Rationale and Source parameters should ideally support Priority parameter specifications.

Example:
Usability:
Scale: <Speed of mastering> defined [Tasks] by defined [Staff Type].
Fail [USA, Task = Query Handling, Staff Type = Customer Service]: 35 hours.
Fail [Europe, Task = Query Handling, Staff Type = Junior Management]: 25 hours.
Goal [Australia, Task = Query Handling, Staff Type = Customer Service]: 30 hours.
Priority [Usability]: Australia Before Europe Before USA <- Marketing Plan 6.5.
Rationale: Past bad experiences with current system.
Source: Technical Director.
Without the explicit 'Priority' statement we would normally prioritize the Fail levels. However, the Priority specification means we should use scarce resources on the Australia Goal before we use them on the two Fail levels. We should then use resource on Europe before the USA.

Related Concepts: Level *337; Risk *309; Gap *359; Dependency *189.

Procedure — Concept *115

A procedure is a repeatable description to instruct people as to the best-known practice, or recommended way, to carry out the task of a defined process. A procedure is part of a process description.

"No matter how many theorists have advocated a procedure, if the procedure has been given a thorough trial and then abandoned, there is a strong presumption that it is unsound."
<-(Mintzberg 1994 Page 135, quoting R. N. Anthony, 1965, Page166, in Planning and Control Systems, Graduate School of Business Administration, Harvard University).

Notes:
1. The procedure description can be kept and presented in various forms including {forms, guidelines, diagrams, video and audio; even unwritten, but understood procedures}.
2. A procedure describes a task that there is benefit in standardizing (usually, because it is carried out so often, to ensure best practice and to prevent error).

Related Concepts: Task *149; Process *113.

Process — Concept *113

A process is a work activity consisting of:
- an entry process, which examines entry conditions
- a task process, which follows a procedure defining the task. There might also be an associated verification process, such as test or quality control
- an exit process, which examines exit conditions.

Processes transform inputs to outputs, using resources, and display their own performance and resource (cost) characteristics.

Notes:
1. A process is a set of actions, carried out by people, nature or machines (agents), which can be defined by inputs, outputs, a sequenced set of actions and its performance, and resource attributes.
2. Processes can only be fully understood by including information about the agent who executes the process (their work environment, competence, experience, training).

Process: "A system of activities that use resources to transform inputs to outputs."

Source: ISO 9000, 2000.

Synonyms: Work Process *113.
Drawn Icon: A rectangle with up arrow on left side.
Related Concepts: Task *149.

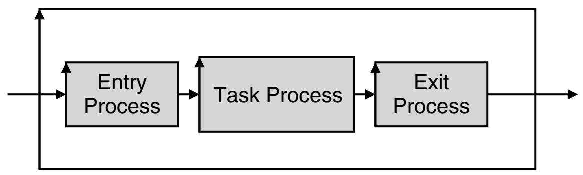


Figure G15
A process with its three main component sub-processes.

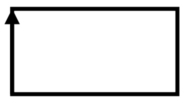


Figure G16
The drawn icon for Process *113.

Process Improvement **Concept *114**

Process improvement is systematic activity, which consists of:

- determining the root systemic causes of human work process problems, then
- changing defined and stable work processes, with the intention of eliminating, or reducing, the human tendency to commit errors, and therefore improving productivity or costs.

'Root causes' are the earliest defect causes in the chain of all causes and effects. It pays to remove the source of a problem. 'Systemic causes' are those that are built into the process, and bound to cause a problem regularly if not improved. The proof of improvement lies in the results from a changed process.

Related Concepts: Root Cause *263; Systemic *262.

Qualifier **Concept *124**

A qualifier is a defined set of conditions embedded in, or referenced by, a specification. All of its conditions must be 'true,' for that specification to apply. A qualifier defines any interesting set of *specific* time, location and event conditions (also known as qualifier conditions). These are sometimes called 'when,' 'where' and 'if' conditions.

Square brackets around the qualifier conditions are used to denote a qualifier. An alternative is to use the Qualifier parameter (which also happens to use square brackets!).

Example:
Goal [Germany, Teachers]: 65%.
Where 'Germany' and 'Teachers' are each qualifier conditions (defined elsewhere). The set of qualifier conditions, and the square brackets, form the qualifier.

Example:
Goal [German School]: 65%.
German School: Qualifier: [Country = Germany, User = Teachers].
Where 'German School' is a defined reusable qualifier. Tagging a 'Qualifier' parameter or statement allows us to simplify a large set of qualifier conditions, and perhaps to describe or summarize them at the same time. 'German School: [Country = Germany, User = Teachers].' would be the logical equivalent to the 'Qualifier:' statement above. You can tag a qualifier directly.

Notes:

1. Qualifiers can be present in any specification (for example, a system attribute specification, a requirement specification, a design idea specification or an Evo step specification). Most Planguage parameter specifications are subject to qualifiers (either explicit, implied or inherited).
2. Any number of qualifier *conditions* can apply to a given specification, expression or statement. There can be multiple instances of any one class of qualifier conditions. There is no sequence requirement for the conditions.
3. Qualifiers are always specified as sets within square brackets, '[ ]', even when a 'Qualifier' parameter is used.
4. Qualifiers have to be evaluated to see if they are 'true' in any given instance. In the case of evaluating an Evo step, this may be done in real time.

5. Qualifiers have the effect of 'dividing up' a system into separate, maybe overlapping, system dimensions or 'views.' This has many uses. One use is to divide up a system into smaller distinct areas for delivery of Evo steps.
6. A qualifier is a substantial contribution to understanding the *priority* of a specification.

Example:
Project Manager Attendance:
Goal [By = Next Year, Market = London, Activity = Customer Meetings, Event Condition = If Sale Agreed]: 90%.
The qualifier conditions specify when, where, during which activity, and after which event – the Goal has validity – and thus has priority over any specification that is not yet *valid.*
MOP:
Scale: % Uptime.
Goal [QQ]: 99.5%.
Stretch [QQ]: 99.9%
QQ: Qualifier [By End of Year, Home Market, Consumer Goods, If Fierce Competition on Price].
Authority [MOP]: Product Planning.
The MOP requirement has two distinct priority mechanisms. The MOP Goal statement has priority over a corresponding ('has same qualifier') Stretch statement. Secondly, the QQ tagged qualifier has a number of qualifier conditions that must all be true for either of the target levels to be in force. The Authority statement gives additional prioritization information for MOP, in relation to other requirements with different powers behind them.

7. A qualifier defines the set of conditions, which together enable, or 'activate', a related statement. The potential qualifier conditions can be *roughly* classified as:
   - **time** conditions: '*when*': Dates, deadlines, relative times to events, weekdays, hours (time spans or precise hour)
   - **place** conditions: any notion of '*where*': Geographical location; type of person, group or role (like trainee, teenager, teacher); system component (like module, program, laptop screen, software, contracts, standards)
   - **event** conditions: any notions of '*if* ': Any occurrence conditions such as:
     - activity commenced or terminated (like project started, policy issued)
     - activity in progress (like testing being carried out, parliament in session, voting in progress)
     - specific indicator set (like red light is on)
     - specific status attained (like Approved, Checked).
   - We can optionally preface event conditions with the logical 'If' parameter in order to emphasize that we must analyze the status of the event to determine if the qualifier condition is 'true'.

Example:
Fail [Europe, Year = After Ten Years, Peace]: 60% ± 20% <- Annual Plan Section 6.4.5.
The three qualifier conditions must all be 'true' for the '60%' requirement level to be a valid requirement. 'Peace' is an example of an event condition. Europe is a place condition.

8. The implication of the qualifier definition, that all qualifier conditions must be 'true' to have effect, is that in a qualifier like [A, B, C], 'A *and* B *and* C' must be valid for the qualifier to 'enable' its related statement.
9. Here are some notes on how the concepts of qualifier, qualifier condition and scale qualifier relate to each other.
   Qualifier = [qualifier condition 1, qualifier condition 2, . . . qualifier condition n]
   Scale Qualifier = [sets up a need for *one* qualifier condition to be specified on use of the Scale]
   Scale Variable = a value for a qualifier condition satisfying a scale qualifier in a statement referring to the defined Scale..
   Scale: . . . . [scale qualifier 1] . . . . . . [scale qualifier 2] . . . . . . [scale qualifier n].
   Parameter [Qualifier]: assigned numeric value.
   Remember a Qualifier is a set of qualifier conditions:
   Parameter [{qualifier conditions}]: assigned numeric value.
   Qualifier = [qualifier condition 1 = scale qualifier 1 = scale variable 1,
   qualifier condition 2 = scale qualifier 2 = scale variable 2,
   . . . ,
   qualifier condition n = scale qualifier n = scale variable n,
   other qualifier conditions as needed not related to the Scale].
   Scale: . . . . . . [scale qualifier n: Default = scale variable n].

Related Concepts: View *484; Time *153; Place *107; Event *062; Condition *024; Scale Qualifier *381; Indicator*195; Status *174.
Keyed Icon: [ ] "Square brackets around any set of qualifiers."

Quality Concept *125

A quality is a scalar attribute reflecting 'how well' a system functions.

Example:
Quality: Includes: {Availability, Usability, Integrity, Adaptability, and many others}.

Notes:
1. A quality is a system performance attribute. All systems have a large number of quality attributes in practice. In a given situation, only the relevant quality attributes will be specified: these are the qualities specifically valued by the stakeholders.
2. All qualities can be described numerically using a defined Scale, or set of Scales. Existing quality levels can be specified as benchmarks, and needed future quality levels can be specified as targets and constraints.

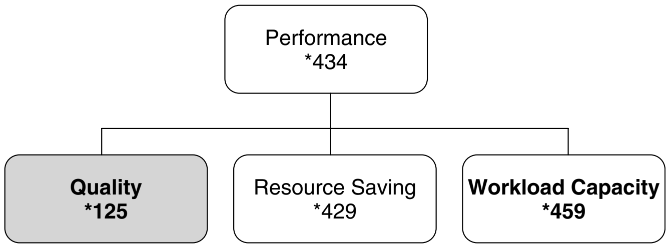


Figure G17
Quality viewed in the context of Performance.

3. Quality is distinct from the other performance attributes: Work Capacity and Resource Saving.
4. **Quality** is characterized by these traits:
   - Quality describes *'how well'* a function is done
   - Quality is *valued* to *some* degree by *some* stakeholders of the system
   - *More* quality is generally *valued* by stakeholders; especially if the increase is free, or at lower cost, than the value of the increase
   - Quality attributes can be *articulated* independently of the specific means (designs) used for reaching a specific quality level – even though all quality levels *depend* on the particular designs used to achieve them
   - A specific quality can be a described in terms of a *complex* concept, consisting of multiple complex and/or elementary quality concepts
   - Quality is *variable* (along a definable scale of measure: as are all scalar attributes)
   - Quality levels are capable of being specified *quantitatively* (as are all scalar attributes)
   - Quality levels can be *measured* in practice
   - Quality levels can be traded off to some degree; with other system attributes valued more by stakeholders
   - Quality can never be perfect in the real world
   - There are some levels of a specific quality that may be outside the state of the art at a defined time and under defined circumstances
   - When quality levels increase towards perfection, the resources needed to support those levels tend towards infinity

Related Concepts: Performance *434; Resource Saving *429; Workload Capacity *459; Quality Requirement *453.

Quality Level — Concept *360

The quality level of a specification is a measure of its conformance to any specified relevant standards.

The Quality Level parameter is used to specify the estimated defect density for a specification: in other words, the number of estimated remaining major defects/(logical) page.

Synonyms: Major Defect Density *360; Specification Quality Level *360.

Related Concepts: Status *174; Remaining Major Defects/Page *060; Specification Defect *043; Major Defect *091; Logical Page *103.

Quantify, To — Concept *385

To quantify is to specify numerically.

Notes:

1. To articulate a variable attribute using a defined scale of measure and specifying one or more numeric levels on the Scale. The resulting specification is 'quantified.'
2. In particular, we need to be clear that 'to quantify' is not identical to the concept of 'to measure.' Measuring is the act of determining where we are, on a defined scale of measure, in practice by using a Meter.

   Quantification is a type of specification that is a prerequisite for other processes, such as Estimation, Measurement, Testing, and Feedback Control.

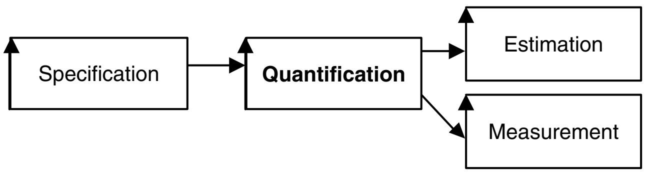


Figure G18

3. Quantification must not be confused with estimation, either. You can quantify without estimation and without measurement! Estimation is to determine a particular (qualifier specified) past or future number, on a defined scale of measure.
4. In Planguage, quantification begins with the definition of a Scale. Quantification continues, and takes on more specific meaning by using benchmarks, targets, assumptions and the many other specification parameters that relate to the defined scale of measure.

Dictionary Definition of 'Quantify':

"1. To determine or express the quantity of; indicate the extent of; measure
2. To express in quantitative terms, or as a numerical equivalent logic to make the quantity or extension of a term or symbol clear and explicit by the use of a quantifier, as all, none, or some."
<- Webster's New World^{TM} College Dictionary *(Third Edition).*

The Planguage definition of quantify (identical to variant 2 above) does not admit to the term 'measure' which is contained in variant 1 of this definition, and in common use of the word. In Planguage, 'measurement' (as in variant 1) is a quite distinct activity, based on the quantification (but not identical to it).

Related Concepts: Scale *132; Meter *093; Measure, To *386; Estimate, To *059.

Range — Concept *552

A range is the extent between and including two defined numeric levels on a scale of measure.

Notes:

1. Range captures a snapshot of some common scalar attribute ranges (using the target and constraint levels). Some examples of different range classifications include:
   - Success Range – project or product success
   - Acceptable Range – not success, but not failure
   - Failure Range – some degree of problems or failure
   - Catastrophe Range – forget it! No use and no hope!
2. A range is usually *implied* by specification of the relevant levels. For example, a performance success range goes from the applicable Goal level in the direction of 'better' forever, unless bounded by an *upper* Survival level.
3. Range, like a level, is interpreted with regard to any specified qualifier conditions. In other words if a scalar level qualifier, for example for a Goal level, is not valid, the range implied by that Goal level is not valid either.
4. Different stakeholders can specify levels with implied ranges that overlap with each other. 'One man's meat is another man's poison,' as the

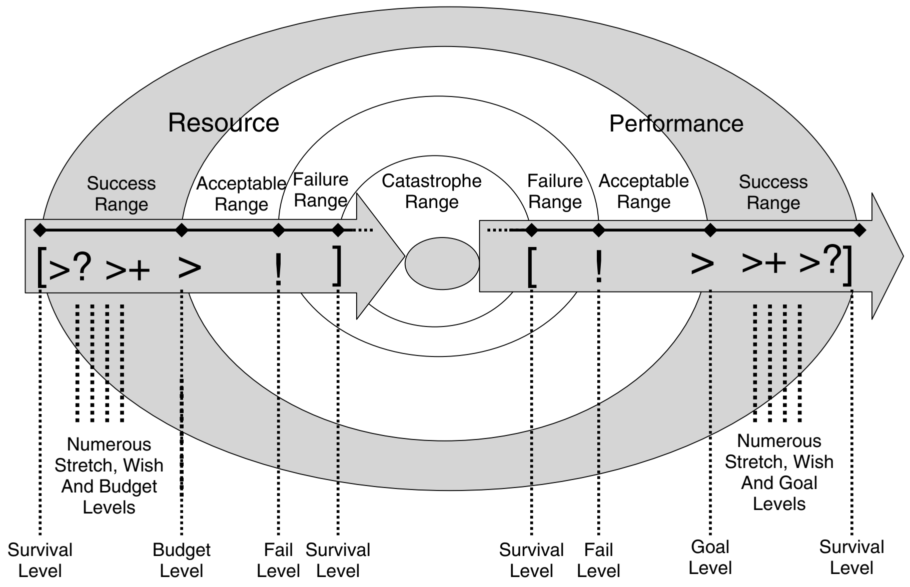


Figure G19
The 'doughnut' diagram indicating different range concepts.

saying goes. Designers need to consider the set of overlapping ranges in an attempt to maximize design efficiency. This is complex, but the problem can at least be clearly seen.

5. A range can include the idea of a set of benchmark levels over time and/or space. For example, a quality level range in a defined market for defined types of people for a defined task during annual periods.
6. A range can describe a gap between a benchmark and a target (the gap is the unfulfilled requirement).
7. A range can describe the change in attribute level as a result of carrying out an Evo step.

Related Concepts: Landing Zone *605; Gap *359; Success Range; Acceptable Range; Failure Range; Catastrophe Range.

Rationale — Concept *259

A rationale is the reasoning or principle that explains and thus seeks to justify a specification. 'Rationale' is a parameter for declaring information that justifies a specification.

Notes:

1. The information can concern the logic, the politics, the economics or whatever is of interest to declare in order to explain and justify a specification.
2. The purposes of a rationale are:
   - to answer questions that readers would ask of a specification
   - to motivate and convince readers about a specification
   - to set up information for risk analysis (is the given rationale true, and will it be later?)

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

Example:
PGB: Goal [UK]: 99.9% <- Annual Plan.
Authority: Board of Directors, Jan 25th.
Rationale [PGB]: Competition in UK prior to new EU Laws about competition.
Basis: Our long-range plan to be the <biggest> in all European countries.
In the example above, the PGB tag is inserted to show how to tie any Rationale statement to another specific statement or statements. This format can be used irrespective of where you specify the Rationale statement. It does not have to be just below or in the immediate vicinity. The Authority and Basis statements are implied to be related, because they are just below the PGB statement.
Note. 'Basis' is quite different from Rationale. Rationale is a set of conditions leading to a desire to make a specification. It explains how we got to that specification. Basis is a specified set of assumptions that underlie a specification. If the basis conditions are changed, then the specification may no longer be valid.

"Theirs is but to reason why: The value of recording rationale."
<-(Hooks and Farry 2000 Chapter 8)

Synonyms: Justification *259; Reason *259.
Related Concepts: Basis *006.
Historical Note: 'Rationale' is a term I found used by Synopsys, CA USA 1996.

Readership — Concept *295

The readership (or intended readership) of a specification is all the 'types of people' we intend shall read or use the specification.
Notes:
1. The 'readership' should be explicitly stated within document header information or other appropriate place. Experience has shown that allowing people to guess what the intended readership is will lead directly to not satisfying some important types of readers.

Rationale:
- It helps authors decide on document content and what terminology to use. For example, which abbreviations and terms might be understood or misunderstood.
- It also helps checkers decide if the writer has communicated clearly and unambiguously with their intended audience.

Example:
Readership: {All Employees, Customers, Suppliers, Contractors, Auditors}.
Synonyms: Intended Readership *295.

Record — Concept *127

A Record parameter is used to inform us about an interesting extreme of achievement. A Record specification states a benchmark level on a defined Scale under specified conditions [time, place, event] for a scalar attribute that represents an impressively good level, or state-of-the-art.

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

Example:
Usability:
Ambition: More user-friendly than <competitors' products>.
Scale: Average Learning Time for Slowest Learners in User Population.
Trend [Main Competitor, Next Year]: 1 minute.
Record [Last Year, UK]: 2 minutes <- Industry Statistics [Last Year].
Fail [New Product, Next Year]: 50 seconds.
Goal [New Product, Anytime]: 20 seconds.
It is good practice to indicate the source (<-) of Record information (as with all scalar level specifications).

Rationale: Record is usually specified to demonstrate that such a level is technically possible under certain specified conditions, and to challenge us to strive to avoid, approach, meet or beat that level, as appropriate.
Levels approaching state-of-the-art are useful to specify, because they tend to be costly and high risk.
Synonyms: Record Level *127.
Related Concepts: Benchmark *007.
Keyed Icon: <<
"The arrow points towards the 'past' as in the Past benchmark, '<' but doubled for emphasis to show that this is an extreme benchmark. Usually used in the context of a scalar arrow (<---<<---O---<<--->)."

Relationship **Concept *142**

A relationship is a connection between objects.
Related Concepts: Object *099; Interface *194; Hierarchical *083; High-Level *082; Supra *264; Downstream *052; Upstream *291; Set *133; Subset *222; Is Part of *621; Consists Of *616; Includes *391; Kin *353; Sibling *265; Kid *266; Parent *267; Dependency *189: Synonym is Depends On; Impacts *334; Is Impacted By *412; Is Supported By *414; Supports *415.

Requirement **Concept *026**

A requirement is a stakeholder-desired, or needed, target or constraint. Within Planguage, requirements specifically consist of vision, function requirements, performance requirements, resource requirements, condition constraints and design constraints.
Given below are some IEEE definitions:

> *Requirement:* "a condition or capability needed by a user to solve a problem or achieve an objective."
>
> *<- IEEE 610.12-1990.*

Example of an IEEE definition of 'requirement': the term 'user' is probably not broad enough to capture the scope of all stakeholders. Another danger of this definition is that it inadvertently includes all designs, and so does not successfully made a sufficiently clear distinction between requirement and design.

> *Requirements:* "Statements, which identify the essential needs for a system in order for it to have value and utility. Requirements may be derived or based upon interpretation of stated requirements to assist in providing a common understanding of the desired operational characteristics of a system."
>
> *<- IEEE P1220. IEEE Standard for Systems Engineering, Preliminary, 1993 in (SEI CMMI 1995).*

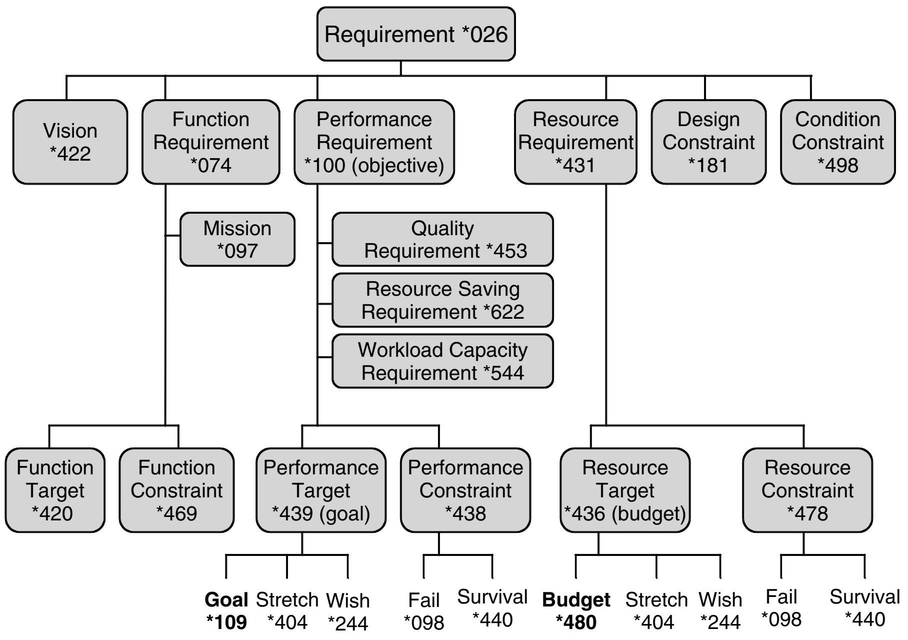

Requirement *026
Vision *422
Function Requirement *074
Performance Requirement *100 (objective)
Resource Requirement *431
Design Constraint *181
Condition Constraint *498
Mission *097
Quality Requirement *453
Resource Saving Requirement *622
Workload Capacity Requirement *544
Function Target *420
Function Constraint *469
Performance Target *439 (goal)
Performance Constraint *438
Resource Target *436 (budget)
Resource Constraint *478
Goal *109
Stretch *404
Wish *244
Fail *098
Survival *440
Budget *480
Stretch *404
Wish *244
Fail *098
Survival *440

Figure G20
Requirement Concepts.

Example of an IEEE definition of 'requirements': better than the previous 1990 standard, as all designs are no longer 'in the picture.'

Notes:

1. Stakeholders should determine their own requirements; they certainly should be involved in discussions about the relative values, costs and priorities of their requirements.
   A systems engineer may be needed to specify the requirements in a suitable way for use by systems engineering projects. Also, there may be a need for building, aggregating and analyzing a set of project requirements across a range of disparate stakeholders.
2. Not all requirements initially specified are necessarily accepted for actual delivery: some requirements may not ultimately be feasible or economic. The key practical idea is to try to identify, and give priority to, the most critical or most profitable stakeholder requirements.
3. A requirement is an input to a design process. Requirements give information to the designer about the necessary nature of their design. A design, whether a specification or an actual implementation, can be judged (using such means as Specification Quality Control (SQC), test, Impact Estimation tables, evolutionary step feedback, or operational use) in terms of how well it satisfies the requirements.
4. Requirements, at different levels of abstraction, can be viewed as inputs to a defined level of design process. In a series of systems engineering processes, one engineer's output ('design,' 'architecture') may become another engineer's inputs or 'requirements'. The conclusion of this is that specifications, no matter what we name them, are requirements only when they are used as input to such a systems engineering process.

So, we need to explain a particular concept of 'requirement' by first specifying the systems engineering process type, or level at which they apply. For example, stakeholder value analysis, product line architecture or component engineering.

5. Requirements can have different levels of priority. Priority is conveyed in a variety of ways, for example:
   - for scalar attributes, by stating any relevant constraint levels using {Fail, Survival}
   - for binary attributes, by stating any relevant constraints using Constraint parameters
   - for scalar attributes, by setting relevant target levels using {Goal/ Budget, Stretch, Wish}
   - by specifying different qualifier conditions for [time, place, event] (Alternatively known as [when, where, if])
   - by specifying other parameters, like Authority, Dependency, Priority and Risk, which provide additional information
6. Requirements are usually assumed to be written in requirement specifications. However, many documents which have 'requirements' in their title may contain little or no *real* requirements: it is unfortunately commonplace to find a high content of design specification for unspecified or vaguely specified requirements. Frequently, the most important, high-level requirements are not stated clearly at all. You must be prepared to look for requirements in other documentation and to ask questions.
7. A design idea is *not usually* a requirement at the *same* systems development process level as the requirements it was designed to satisfy. However, once a design idea is 'fixed' at a project development process level, it becomes a 'design constraint' (which is a requirement type) for the next level. In other words, a design idea usually becomes 'valid as a requirement' at the next level of the development process, after the level it was 'designed.' It is then a valid 'requirement' for all future 'lower' levels.

Example:
Usability:
Type: Quality Requirement.
Scale: Time to learn [defined Task].
Goal [By Initial Delivery, CW-Task]: 30 minutes. "A simple requirement goal."
CW-Task: Defined As: <mastering> <frequent tasks> at the <standard workstation>.
Interface:
Type: Design Idea.
Specification: The computer terminal interface must look exactly like our old one.
Impacts: Usability. "A design to meet the Usability requirement."
Usability is a strategic requirement. Interface is a design idea that we hope will satisfy the planned level for Usability. Once adopted, Interface is handed to others in the development process, such as estimators, constructors and testers, and is then classed as a design constraint, at this lower level.

Related Concepts: Requirement Specification *508; Need *599; Problem Definition *598; Problem *270; Target *048; Constraint *218; Stakeholder *233; Objective *100.

Requirements Engineering **Concept *614**

Requirements Engineering is a requirements process carried out with an engineering level of rigor. Requirements Engineering includes all aspects of requirements gathering and maintenance, including but not limited to:

- requirements solicitation (including stakeholder analysis)
- requirements analysis
- requirements quality control
- requirements review
- requirements change management
- requirements specification (the process)
- contracting and bidding requirements
- requirements risk analysis
- requirements priority analysis.

Synonyms: Requirement Engineering *614.

Related Concepts: Requirement *026; Requirement Specification [Process] *634.

Requirement Specification [Specification] **Concept *508**

A requirement specification is a defined a set of requirements. It also includes any relevant requirement background, such as benchmarks, and also any appropriate commentary.

Note:

1. A requirement specification is the output of a requirement specification process, which is a subset of a requirement engineering process.

Template: Requirement Specification Template.

Synonyms: Requirements Specification *508.

Related Concepts: Requirements Engineering *614; Requirement Specification [Process] *634; Requirement *026; Specification *137; Commentary *632; Background *507; Core Specification *633.

Resource **Concept *199**

A resource is any potential system input. A resource is any kind of input 'fuel' necessary for building, operating or maintaining a given system. A resource is an asset or a supply that can be used to produce the functionality or performance levels of a system.

A resource can be defined using a scale of measure. A requirement for a resource can be specified by a target or a constraint level. Previous levels of resource utilization (costs) can be specified by benchmark levels, like Past.

We can distinguish between budgeted and unbudgeted resources. Resource budgets are found in our formal plans.

Notes:

1. The emphasis is on the concept of '*potential*' resource. A *potential* resource is the total resource that might *theoretically* be consumed, used, applied or produced. This is in contrast to a *level* of that resource that we *plan* to use, called a Budget (a resource target).
2. We should not plan to use more than the resource actually available at any point in time, place or circumstances. However, when one type of resource is *unavailable*, we can consider the possibility of employing *another resource* to achieve our aims; this is one kind of tradeoff. The classic example is 'time versus money.'

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

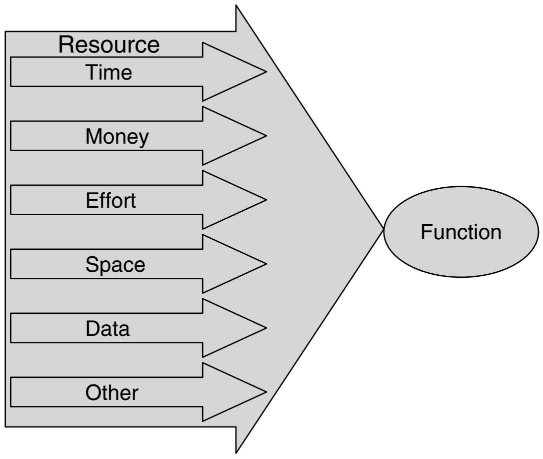


Figure G21
Arbitrary examples of some system resources.

3. Resources must be viewed with regard to the 'total potential resource available – now and later,' under the defined conditions. For example: 'in our divisional budget,' 'within the market window' or 'by the contract handover deadline.'
4. When you plan to limit the use of specific resources, you do so by setting a resource target (for example, 'Budget: €2 million.') or a resource constraint (for example, 'Survival: €2.2 million.').
   You might also specify a global or policy constraint (see example below).
   Example:
   Innovation Constraint [Division A]:
   Type: Constraint [Financial Resource].
   Scale: % of the annual research budget.
   Goal: 20%.
   Authority: Divisional Manager.
5. Resources that are input to a system differ from resource savings. Resources consumed are the costs of developing and operating a system. A resource saving is a relative reduction in consumption of a resource once a system is operational. For example, systems engineering effort as an input resource can be applied to save system user learning time (a resource saving).
6. Common usage of the term 'resource' in the United States (USA) is to mean 'people.' The Planguage definition is far wider than this.

Related Concepts: Resource Requirement *431; Resource Target *436; Resource Constraint *478; Benchmark *007; Target *489; Constraint *218; Cost *033; Resource Saving *429.

Keyed Icon: --->O "A scalar attribute arrow into a function oval. A simplified alternative is '–O'."

Resource Constraint — Concept *478

A resource constraint is a resource requirement, which specifically *restricts*, or *serves as a warning* about, the level that can be used of a resource.

A resource constraint is specified as a scalar resource attribute level. It signals the level at which *some degree of system failure* will be experienced or the level at which the entire system becomes threatened.

Two main parameters can be used to specify these constraint levels: Fail (Fail and worse might be recoverable) and Survival (worse is unrecoverable).

Notes:

1. Resource constraints are imposed or suggested by defined stakeholders. These stakeholders and their reasons should be explicitly documented with the constraint level, for example, by using Authority, Source, Rationale or Stakeholder parameters.
2. The Fail and Survival concepts are adequate for most scalar constraint purposes. However, Catastrophe is also available for use. It is a matter of taste.

Related Concepts: Constraint *218; Performance Constraint *438; Resource Target *436; Fail *098; Survival *440; Catastrophe *602; Range *552: See Failure Range and Catastrophe Range.

Example:
Memory Space:
Type: Resource.
Scale: Gigabytes of defined [Memory Component].
======================= Targets =======================
Wish [Memory Component = Online Backup]: 1,000 Gb <- Design Team.
Rationale: Improves Recovery Speed.
Stretch [Memory Component = Online Storage, US Market]: 500 Gb? <- Marketing.
Budget [Memory Component = Primary]: 100 Gb <- Initial Software Size Estimates.
======================Constraints ======================
Fail [Memory Component = Online Storage, US Market]: 250 Gb Or Less? <- Marketing.
Rationale: Large Scale Users must have this level <- US Sales.
Survival [Memory Component = Online Storage, US Market]: 100 Gb? <- Marketing.
Rationale: Nobody would even consider our system with less <- US Sales.
Some examples of Resource Constraint specification.

Resource Requirement — Concept *431

A resource requirement specifies how much of a resource should be made available for later consumption. A resource requirement is a scalar attribute with one or more resource targets and/or resource constraints specified.

Example:
Maintenance Expenditure:
Scale: Million € annually.
Budget [First Four Years Average]: 3 million €.
Fail [Any Single Operational Year]: 4 million € <- Client limit in contract §6.8.

Example:
Development Effort:
Scale: Engineering Hours applied to the contract.
Wish [Europe Release]: 10,000 hours <- Project Manager.
Survival [All Releases]: 30,000 hours <- Maximum corporate availability by deadline.
An example showing a resource target and a resource constraint specified for each resource requirement.
Related Concepts: Resource *199; Resource Target *436; Resource Constraint *478.

Resource Saving **Concept *429**

A resource saving is a performance attribute of a system. It expresses 'how much' better the system currently performs in terms of resources than it did at some previous benchmark time.
For example, a resource saving can express how much less resource in training effort or maintenance cost is needed in one system compared to another. This measure can be used for benchmarking or for setting requirements, or even for reporting on progress in design or actual measured implementation of new systems.
Synonyms: Saving *429.
Related Concepts: Performance *434; Quality *125; Workload Capacity *459; Resource Saving Requirement *622.

Resource Target **Concept *436**

A resource target is a budget. It is a scalar requirement; a resource level we aim, or might possibly aim, towards keeping within while working towards achieving the other requirements.
Three parameters are used to specify resource targets {Budget, Stretch and Wish}. A Budget level is the *primary* resource target type. A Stretch level represents a resource target that is *not* committed, but is a level for challenge and motivation. The Wish level represents a resource level that would have value to some stakeholder, but again is not committed.
Resource targets represent the reasonable, perhaps profitable, levels of cost, we must expect to pay to reach our performance and function targets within any constraints. They differ from resource *constraints*, which are the levels that signal problems, danger or lack of profitability. We do not plan and design to merely stay *within* resource constraints, but to *avoid* going anywhere near them at all!

Example:
Memory Space:
Scale: Gigabytes of defined [Memory Component].
Wish [Memory Component = Online Backup]: 1,000 Gb <- Design Team.
Rationale: Improves Recovery Speed.
Stretch [Memory Component = Online Storage, US Market]: 500 Gb? <- Marketing.
Budget [Memory Component = Primary]: 100 Gb <- Initial Software Size Estimates.
Synonyms: budget *436 (with a small 'b' to distinguish it from the parameter, 'Budget').

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

Related Concepts: Budget *480; Resource *199; Target *048; Wish *244; Stretch *404; Resource Constraint *478; Resource Requirement *431; Requirement *026.

Result Cycle **Concept *122**

Within Evo, a result cycle is an entire Evo step cycle aimed at delivering a result that moves towards satisfying the overall requirements.

Notes:

1. A result cycle is a cycle consisting of 'Plan-Do-Study-Act' activities.
2. It can involve any kind of system change, small or large: for example, factory production modification, software program alteration, organizational restructure, new software product development and design of new businesses.
3. A project using Evo will execute numerous result cycles. The emphasis is on 'contact with reality' and *using consequent feedback to adjust.*
4. A result cycle consists of:
   - a strategic management cycle: A strategic management cycle is concerned with controlling and monitoring the overall change process. Amongst other things, a strategic management cycle approves proposed changes against strategic objectives and adopted high-level strategies. It co-ordinates with other programs and projects. It acquires development budgets. It analyzes feedback measurements. It decides the next step.
   - an implementation cycle: Implementation means 'taking a plan, or an idea, and turning it into reality.' An implementation cycle consisting of the following sub-cycles:
     – a development cycle: This is an optional backroom cycle concerned with acquiring/purchasing and developing, any products required for the production and/or delivery cycles. For example, any new systems development would be carried out within this cycle.
     – a production cycle: This is an optional backroom cycle concerned with product integration, or manufacturing and distribution of any products required for the delivery cycle.
     – a delivery cycle: This is the actual delivery of the deliverable to the use. In other words, a delivery cycle contains the initial operational implementation of an Evo step, and its handover to stakeholders. It involves implementation activities such as training, installation and field-testing. The type and size of system change involved in a delivery cycle can vary, but is usually subject to project-defined step constraints on resource utilization. Usually, both financial cost and delivery frequency must be between 2% and 5% of project total budgets for cost and time respectively.
5. Result cycles, for different steps, can be executed serially and in parallel. The reason for this is the variable times taken for implementation (specifically development and production cycles) and the Evo requirement to achieve a reasonably short delivery cycle frequency. For example, the average delivery cycle frequency could be stipulated to be weekly or monthly, but a specific result might take six months from initiation to actual result delivery, due to such factors as research cycles, order cycles, construction cycles and approval processes. These processes would normally be sought to be done *in parallel* with other Evo cycle activities, so

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

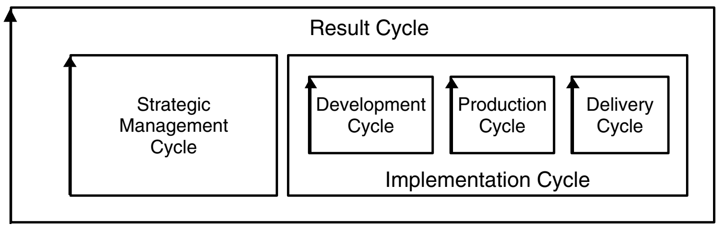


Figure G22
Diagram shows the component cycles of a Result Cycle.

that the Evo management team and their stakeholders would still experience some result delivery within the stipulated delivery cycle time.

6. The development and production cycles are termed 'backroom' activities and the delivery cycle is termed a 'frontroom' activity. One useful analogy is to think of the way in which a restaurant delivers to its customers. Ideally, delivery to the table is independent of food and drink preparation times!

Synonyms: Result Production Cycle; Step Cycle.

Related Concepts: Delivery Cycle *049; Development Cycle *413; Implementation Cycle *123; Production Cycle *407; Strategic Management Cycle *408.

Review — Concept *197

A review is any process of human examination of ideas with a defined purpose and defined standards of inquiry.

Notes:

1. A 'go/no-go review' is a particular type of review for giving approval, or not, to a particular plan or idea.
2. Reviews should always have the benefit of the specification under review having successfully exited SQC processes using both Specification Rules and Specification Review Rules. Such SQC processes determine a specification's objective craftsmanship quality (conformance to standards). For example, a major review with

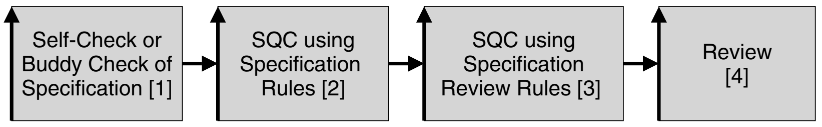

Self-Check or Buddy Check of Specification [1]
SQC using Specification Rules [2]
SQC using Specification Review Rules [3]
Review [4]

Figure G23
The diagram shows Review following SQC processes. The sequence of SQC processes leading to review(s) is as follows:

- Before any SQC is carried out, informal checking might be carried out by the specification writer or by a colleague [1]
- SQC (a more formal team process) is carried out by a group of people checking the specification against specification rules [2]
- If successfully SQC exited, further SQC can be carried out using specification review rules. This is to check the validity of entering a review process by carrying out a number of pre-checks using the relevant review criteria (types of review include Architecture Review and Business Review.) [3]
- If successfully SQC exited, a review can be carried out to decide on future actions [4].

financial consequences should not proceed if the estimated remaining major defects/page for a specification are, at entry, any greater than 1.0.

Related Concepts: Specification Quality Control (SQC) *051; Specification Rules *129; Specification Review Rules *543.

Risk **Concept *309**

A risk is any factor that could result in a *future negative* consequence. A Risk parameter can be used to specify known risks.

Notes:

1. Negative results are results that are worse than required, planned, or expected.
2. Examples of risk factors include:
   - lack of information about a design idea
   - inappropriate information about a design idea.

Source: See Bernstein's book on the history of risk (Bernstein 1996). One prominent economist (Knight) wanted to distinguish risk from uncertainty, in the sense that risk was measurable (Bernstein 1996, Page 219). Knight was also skeptical as to whether past data was sufficiently like a specific unique instance, and sufficiently detailed, to tell us what the probability of a future event would be.

Synonyms: Threat *309.

Related Concepts: Uncertainty *310; Safety Factor *131.

Role **Concept *253**

A role is a defined responsibility, interest or scope for people.

Related Concepts: Role [SQC] *411; Stakeholder *233.

Rule **Concept *333**

A rule is any statement of a standard on how to write or carry out some part of a systems engineering or business process.

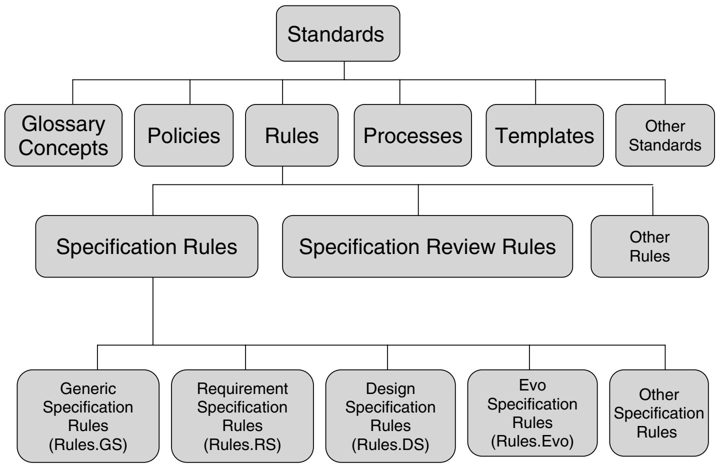

Standards
Glossary Concepts
Policies
Rules
Processes
Templates
Other Standards
Specification Rules
Specification Review Rules
Other Rules
Generic Specification Rules (Rules.GS)
Requirement Specification Rules (Rules.RS)
Design Specification Rules (Rules.DS)
Evo Specification Rules (Rules.Evo)
Other Specification Rules

Figure G24
Rules as standards. Some of the different types of Specification Rules are shown.

Notes:

1. Numerous different types of rule exist, including business rules, system rules and design rules.

Related Concepts: Standard *138; Specification Rule *129; Specification Review Rule *543.

Safety Deviation **Concept *405**

A safety deviation is a measure of the estimated-or-observed difference between a required safety margin, and the estimated or actual system attribute level. 'How safe?' or 'How safe compared to plan?'

Notes:

1. Each scalar attribute target (performance goals and resource budgets) will potentially have its own computable value for safety deviation.
2. Safety deviation expresses how far away the current design proposal, or Evo step implementation, is estimated to be from the desired safety level. The higher the negative deviation is, the greater the 'risk of failure' to deliver the target level of the attribute.
3. The safety deviation can be use by technical management to monitor the progress of a design or a real evolving project delivery. Management will need some policy regarding setting and respecting safety factors. They will need to set some standards regarding the degree to which designs include planned safety factors. This is a specific tactic for risk management.
4. When using the Impact Estimation method and a spreadsheet model, the safety deviation computations can normally be done automatically using the values of the requirements and the design impacts. Automatic warning of insufficient safety is a possibility.

Related Concepts: Safety Factor *131; Safety Margin *637.

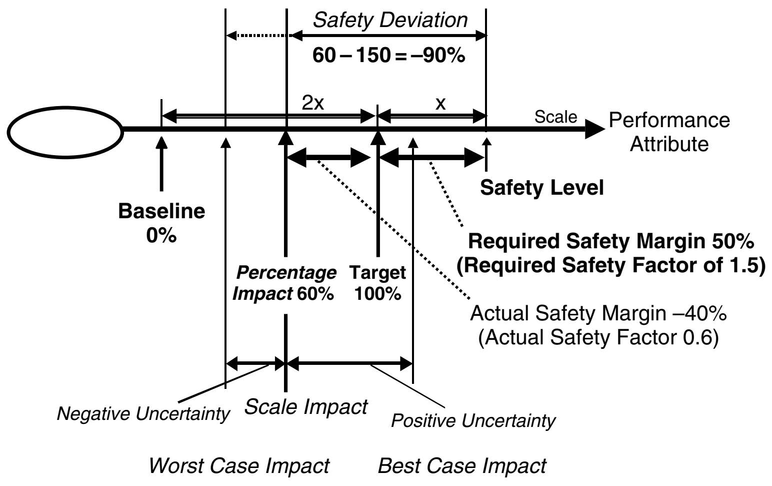


Figure G25

Diagram showing calculation of Safety Deviation for a quality objective. The safety margin is relative to the target level and the distance between the baseline and the target. In this example, as the required safety margin is 50%, it must be 50% beyond the target level. The distance (x) is worked out from the distance between the target and the baseline (2x).

Safety Factor **Concept *131**

A safety factor is the dimensionless ratio of 'conscious over-design' that is either required, or actually applied to some part of a system.

Notes:

1. A safety factor is used to communicate about risk. It is used to ensure that the design compensates adequately for both systems engineering and operational uncertainties.
2. Historically, safety factors were applied to mechanical loads. We are using it here to describe the amount of safety margin we wish to have designed into the system. The target and constraint levels are specified at the required levels and then the safety factor is applied to allow safety margins. (An assumption is being made here that there is only one safety factor involved; there could be several.)
3. A safety factor is either prescribed by standards, such as engineering rules or policy, or it is specified at project level.
4. A safety factor is a dimensionless ratio. Compare to a safety margin, which is either expressed using units of measure (as it is the difference between two levels on a Scale), or as a percentage value based on the required target or constraint level being 100%.

Example:

Required Level	*Estimated/Actual Level*	*Estimated/Actual Safety Factor*	*Estimated/Actual Safety Margin*
100%	100%	1	0%
100%	50%	0.5	−50%
100%	200%	2	100%

This example assumes no specific safety factor has been set. It calculates the estimated/actual safety factor and safety margin based on the required level being 100%.

5. If we want to explicitly specify a safety factor, we can do so in a variety of ways using the Safety Factor parameter.

Synonyms: Safety *131.

Related Concepts: Safety Deviation *405; Safety Margin *637.

Keyed Icon: nX "Where n is the numeric safety factor."

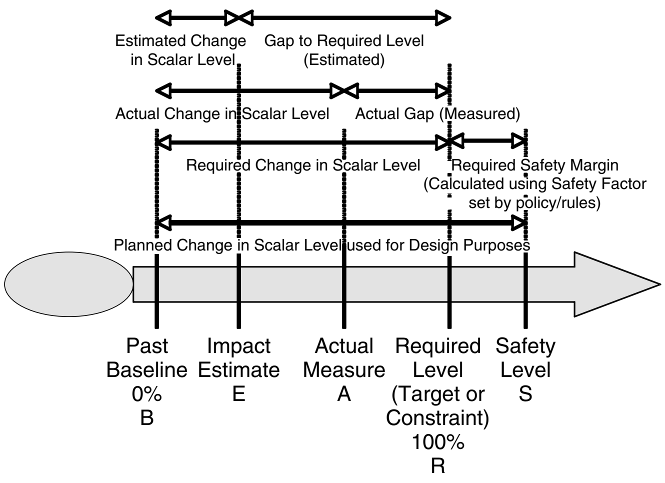
Example:
Safety Factor: 3X.

Safety Margin **Concept *637**

A required safety margin is a scalar difference between a required defined target or constraint level, and its calculated safety level derived using the appropriate safety factor.

Notes:

1. An estimated or actual safety margin can also be calculated. If no specific required safety factor is specified then the safety margin can be calculated relative to the (estimated or actual) target or constraint level (100%, equivalent to a safety factor of 1).
2. A safety margin can also be expressed as a percentage based on the target or constraint level being 100%, and the baseline being 0%.


Estimated Change in Scalar Level
Gap to Required Level (Estimated)
Actual Change in Scalar Level
Actual Gap (Measured)
Required Change in Scalar Level
Required Safety Margin (Calculated using Safety Factor set by policy/rules)
Planned Change in Scalar Level used for Design Purposes
Past Baseline 0% B
Impact Estimate E
Actual Measure A
Required Level (Target or Constraint) 100% R
Safety Level S

3. A 'safety factor', by comparison, is a dimensionless ratio.
Related Concepts: Safety Factor *131; Safety Deviation *405.

Scalar **Concept *198**

Scalar is an adjective used to describe objects, which possess or are measured using at least one scale of measure.

Notes:

1. Performance and resource attributes are scalar.
2. Scalar attributes, provided they are not elementary (one Scale only), can have numerous scales of measure (A complex scalar attribute will possess more than one elementary attribute.)
3. All numeric levels on scales of measure can be described as *scalar* values.
4. A scalar object can be contrasted to a binary object, which is not scalar, but is in one of two states (commonly, either present or absent).

Related Concepts: Scale *132; Binary *249.

Scale **Concept *132**

A 'Scale' parameter is used to define a scale of measure. All elementary scalar attribute definitions require a defined Scale.

A Scale states the fundamental and precise operational definition for a specific scalar attribute. It is used as the basis for expressing many of the parameters within the scalar attribute definition (for example, Meter, Goal and Budget): all scalar estimates or measurements are made with reference to the Scale. The Scale states the units of measurement, and any required scalar qualifiers.

Notes:

1. A Scale is not a numeric level along the defined Scale (it is not a benchmark, target or constraint).
2. A Scale is not the measuring instrument (*that* is specified by the Meter parameter).

   A Scale describes something, which is variable, trackable, observable and countable in nature. A Meter specification is the definition of the *means* of measuring the 'level of capability' expressed by a Scale. For example, a Meter is the 'voltmeter' for measuring on a Scale of 'volts.' A Scale is abstract, while a Meter is a real-world, practical means of obtaining measurements. A specific Meter itself has multiple performance and cost attributes. These are the basis for selecting a particular Meter.

There is a big distinction between units of measure (Scale) and a measuring tool (Meter or Test). Units of measure can be used to express clear ideas, like requirements, quite independently of the possibilities and problems of measurement itself. I can express a clear idea, "I want to get to the moon and back in one second," quite clearly. The fact that I cannot really do it, or measure it, is beside the point. I stress this because I have discovered that many people waste their energy arguing against a particular *quantification*, when all their arguments are only related to the difficulty of its accurate *measurement*.

3. Many Scales are specified as 'generic scales.' A generic scale is a Scale that requires final specification of 'scale qualifiers' (in the Scale definition) by means of 'scale variables' (in a target or constraint specification), in order to have an operationally precise definition.

Example:
Scale: Time to Master defined [Tasks] by defined [Learner Type].
There are two scale qualifiers in the above generic scale definition, which require definition by scale variables. For example, 'Goal [Tasks= Update, Learner Type= Novice]: 30%'.

"To leave [soft considerations] out of the analysis simply because they are not readily quantifiable or to avoid introducing 'personal judgments,' clearly biases decisions against investments that are likely to have a significant impact on considerations as the quality of one's product, delivery speed and reliability, and the rapidity with which new products can be introduced."
<- *R. H. Hayes et al.* Dynamic Manufacturing, *Free Press 1988 NY Page 77,* quoted in Mintzberg (1994 Page 124)

"Aligning Rewards with Measurements 'You have to get this one right. . . . a universal problem: What you measure is what you get – what you reward is what you get. Static measurements get stale. Market conditions change, new businesses develop, new competitors show up. I always pounded home the question 'Are we measuring and rewarding the specific behavior we want?''"
<- *Jack Welch, former CEO* General Electric *(Welch 2001 Page 387)*

Synonyms: Scale of Measure *132.
Related Concepts: Scale Qualifier *381; Scale Variable *446; Meter *094.
Keyed Icon: -|-|-

Scale Impact — Concept *403

For a scalar requirement, a scale impact is an absolute numeric value on the scale of measure. It can be an estimated value, or actually achieved, measured value. It is the level estimated or achieved if a specified design idea (or set of design ideas or Evo step) is implemented.

Notes:

1. In an Impact Estimation table, Scale Impact is customarily used together with Percentage Impact, as alternative views of the impact estimate. We use Scale Impact when we just want to know the real final result, which

includes the effects of implementing all previous designs. We use Percentage Impact when we want to understand the effect in relation to moving from the baseline towards the goal. In other words, Scale Impact is an absolute numeric value, while Percentage Impact is a relative value dependent on the Scale Impact, and the Baseline to Target Pair.

2. Care has to be taken, as the impact of a design idea varies, depending on the system technology it is added to and used in. In other words, the impact of a design idea is not a constant, irrespective of the circumstances it is implemented in. There can be dependencies and interactions. Altering the order of implementing design ideas could affect the immediate level of impact of any specific design idea. However, given that the choice is usually just 'what shall we implement next on a specific system?' it is not necessary to assess the impacts of all the valid design idea combinations.

Synonyms: Absolute Impact *403.
Related Concepts: Incremental Scale Impact *307; Percentage Impact *306.

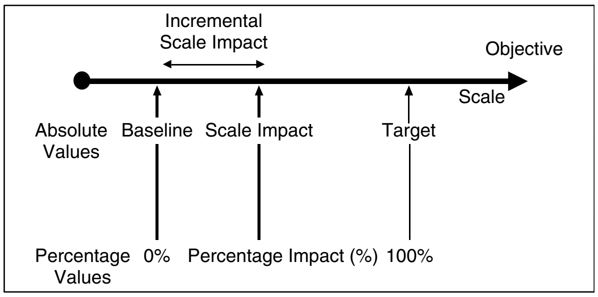


Figure G26

Scale Qualifier — Concept *381

A scale qualifier is a term within the *definition* of a Scale parameter. It specifies the need for a qualifier condition with an assigned scale variable to be specified when referencing or applying the Scale in another statement. For a given Scale, any useful number of scale qualifiers can be defined.

Example:
Scale: Time to learn a defined [Task]. "Task is a scale qualifier."

Scale qualifiers are generic; each scale qualifier needs to be explicitly assigned a corresponding 'scale variable' (unless a default is being used) when the Scale is used in other parameter statements (such as any benchmarks or targets).

Example:
Goal [Task = Setup]: 10 minutes. "Setup is a scale variable defining the scale qualifier, Task that was defined in the previous example."

The purpose of scale qualifiers is to allow a scale specification to be more generalized and flexible; this consequently makes a scale specification more *reusable.*

Notes:

1. A scale qualifier is expressed and specified by enclosing the qualifier condition in square qualifier brackets. The word 'defined' is optionally

specified immediately prior to the square brackets, to help emphasize that a more specific definition needs to be provided when the Scale is referenced, for example by a Goal statement.

Example:
Scale: The defined [Time Units] needed to do a defined [Task] by a defined [Employee Type].

2. A *default option* can be specified in order to make explicit specification unnecessary.

Example:
Scale: The defined [Time Units: *Default = Hours*] needed to do a defined [Task] by a defined [Employee Type].

3. The scale qualifier parameter can also be 'referenced' by using the *same sequence* as used in the scale definition: Note in this example, an additional qualifier condition, not in the original scale definition, has been added. This is OK. You can add any number of additional conditions that you want.

Example:
Scale: The defined [Time Units] needed to do a defined [Task] by a defined [Employee Type].
Goal [Hours, Answering Help Desk Queries, Experienced, Country = Finland]: 60 Hours.

Or, an *explicit* reference to the scale qualifier tag ('Time Units = Hours') may be made, for increased clarity.

Example:
Scale: The defined [Time Units] needed to do a defined [Task] by a defined [Employee Type].
Past [Time Units = Months, Task = Complaint Handling, Employee Type = Supervisor]: 6 Months.

4. The sequencing of scale qualifiers and scale variables is not critical as long as the parameters are unambiguous to the specification user.

Synonyms: Scale Parameter *381; Embedded Scale Qualifier *381.
Related Concepts: Qualifier *124; Scale Variable *446.

Scale Uncertainty — Concept *143

A scale uncertainty is an estimate of the error margins for a specific scale impact (that is, it provides information about the plus-and-minus range on the scale of measure over which an estimate for a scale impact can possibly vary). It allows calculation of the best and worst case borders.

Notes:

1. Experience data should be used for guidance, and specified as evidence together with its source(s).
2. In some cases, the error margins may not be symmetrical about the main estimate. It may be appropriate to use only the more extreme uncertainty value, or to specify the asymmetry directly using the two different numbers (for example, +30% and −40%).

Related Concepts: Scale Impact *403.

Scale Variable — Concept *446

A scale variable is the specific term assigned to 'finally' define a qualifier condition for a scale qualifier.

If we fail to explicitly define a scale variable in a particular statement and there is a defined *default* scale variable, then the defined default (. . . for defined [Tasks: Default = Update] . . .) will be assumed to be the intended scale variable for that statement. If there is no default defined, then the statement, in this instance, is defective.

Example:
Responsiveness:
Scale: Number of Days after a defined [Day] until enquiries answered.
Goal [Day = Monday]: 3.
'Day' is the scale qualifier. Monday is a constant, assigned as the relevant scale variable, out of the set of the possible days of the week.

Example:
Excess Speed:
Scale: Kilometers per hour in excess of the defined [Maximum Speed: Default = Posted Legal Limit Speed].
Fail [Maximum Speed = 80]: 0 Kilometers per hour.
You would have to finally determine the definition in real driving conditions.

Example:
Scale: The time needed for a defined [Task] by defined [People] in defined [Places].
Goal [Update, Naval Officer, At Sea]: 20 minutes.
Or, alternatively, using explicit references to the scale qualifiers, (Task = . . .).
Goal [Task = Update, People = Naval Officer, Places = At Sea]: 20 minutes.
'Update', 'Naval Officer', and 'At Sea' are scale variables, defining one of the three scale qualifiers (Task, People and Places). The scale variables are also specification variables because we don't really know what they mean until we look at their definition. For example, what if 'At Sea: Defined As: In any craft which floats on any type of water'? Does that include wooden model boats in a small pond?
Related Concepts: Scale Qualifier *381; Specification Variable *456.

Scope — Concept *419

A 'Scope' describes the extent of influence of something. Scope can apply to anything, like a specification, or a specified system or project. The 'extent of influence' can be described in any useful terms. This includes using any Planguage expressions or parameters. For example, any [time, place, event] qualifier conditions, and any other parameters, such as 'Stakeholders', can define the extent of influence of a specific specification within the system scope.

Notes:

1. There are two especially useful notions of scope:
   - Global Scope: global scope specifications (potentially) influence or dictate something (like a constraint) to all areas of a defined system, unless some overriding or higher-priority specification cancels its influence. For example, 'Project Scope' (Hooks and Farry 2000 Pages 43–58).
   - Local Scope: a local specification is unable and unwilling to influence or determine specifications (such as requirements and designs) beyond a defined sub-system area.

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

Related Concepts: View *484; Context *483; Qualifier *124; Time *153; Place *107; Event *062.

Example:
Scope [Project X]: USA Parent Market only.
An example of an explicit Scope specification: Scope can otherwise be indicated by a qualifier and by many other parameters (such as Authority for example).

Side Effect — Concept *273

An impact by a design idea, on any requirement attribute, other than the direct impact(s) we primarily intended.

Notes:

1. Side effects can be evaluated at a design stage and/or observed at an implementation stage, or even operational or decommissioning stage. Conventional usage of 'side effect' implies 'negative effects,' but positive side effects can be just as likely, and just as interesting!
2. Side effects can be of the following categories:
   - 'Intended or unintended': 'Intended' means that we have chosen the design because we knew about and valued those particular side effects;
   - 'Known or unknown': 'Known' means we were aware of the existence and possibly the levels of the side effects. 'Unknown' means we were not initially aware of the side effects, but may have become aware of them at some later stage of considering the design (such as in testing, in a review or in operation);
   - 'Negative, neutral or positive'.

Related Concepts: Impacts *334: This parameter is used to specify side effects.

Software Engineering — Concept *572

Software engineering is the discipline of making software systems deliver the required value to all stakeholders.

Notes:

1. Software engineering includes determining stakeholder requirements, designing new systems, adapting older systems, subcontracting for components (including services), interfacing with systems architecture, testing, measurement and other disciplines. It needs to control computer programming and other software related sub-processes (like quality assurance, requirements elicitation, requirement specification), but it is not necessary that these sub-disciplines be carried out by the software engineering process itself. The emphasis should be on control of the outcome – the value delivered to stakeholders, not of the performance of a craft.
2. The concept 'required value' (above) is used to emphasize the obligation of the software engineer to determine the value or results truly needed by the stakeholders, and not to be fooled by omissions, corruptions and misunderstandings of the real-world value.
3. The concept 'all stakeholders' (above) is used to emphasize the broad range of internal stakeholders (like the development project and the producing organization), and external stakeholders (such as users,

customers, governments, add-on suppliers) that the software engineering process must be obliged to deal with. We are consciously trying to break away from older, narrower notions that software engineering is all about satisfying users or customers alone.

Related Concepts: Software *570; Engineering *224; Systems Engineering *223; Stakeholder *233; Value *269.

Source — Concept *135

'Source' is a synonym for process input *information* (as opposed to process input *materials*).

Notes:

1. Source specifications used in SQC, are contained in documents that are usually of earlier production, and probably at higher levels of authority, global scope and abstraction. For example: contracts are sources for requirements. Requirements are a source for design. Requirements and design are sources for Impact Estimation. Design is source for planning and construction or programming. Older specifications and change requests are sources for updated specifications.

Related Concepts: Evidence *063.

Keyed Icon:

Specification — Concept *137

A 'specification' communicates one or more system ideas and/or descriptions to an intended audience. A specification is usually a formal, written means for communicating information.

Notes:

1. A specification is usually written, but it could be oral.
2. The term 'specification' can refer to a single element of a larger specification or to a larger set of specifications. It includes the entire set of parameters and lines of text needed to specify an idea.
3. The specification concept can deal with past, present and future; it is not confined to requirement or design specification.
4. There are many classes of 'specification' including {requirements, design analysis (such as Impact Estimation tables), and project plans (such as Evo plans)}.
5. 'Specification' can be described as a class of document that is used to control the outcome of a project.
6. The term 'specification' is often specifically intended to refer to project specifications, sometimes popularly called 'specs.'
7. Specification can be categorized as Commentary or Non-Commentary. Non-Commentary consists of Core Specification and Background Specification. This categorization recognizes the significance of the specification content. For SQC purposes, it is important to make this distinction, as finding major defects in the Core Specification is the key task.

Abbreviations: Spec.

Related Concepts: Definition *044; Description *416; Documentation *579; Document *180; Planguage Concept *188; Commentary *632; Non-Commentary *294; Background *507; Core Specification *633.

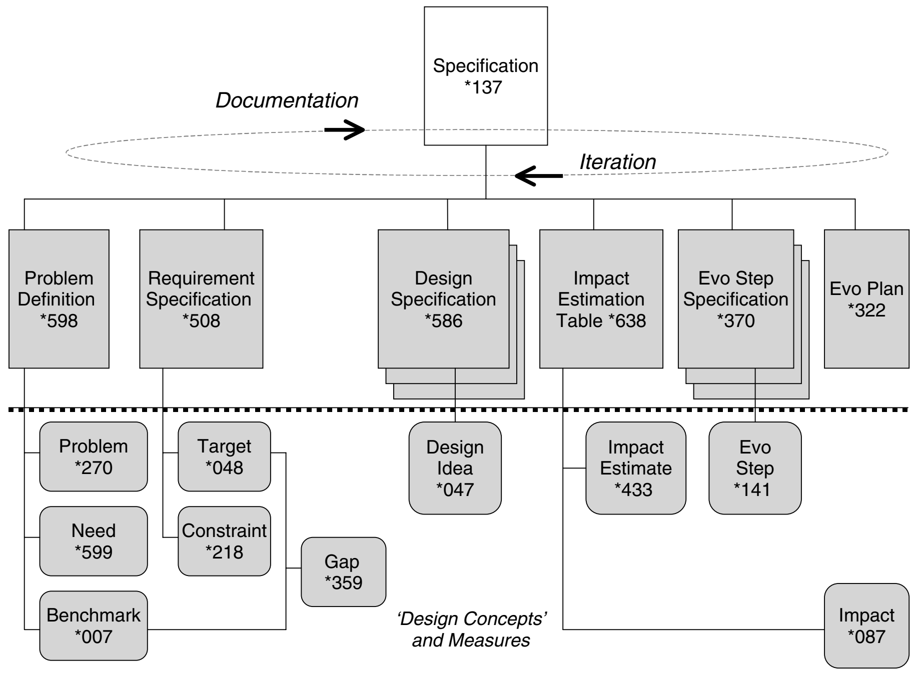


Figure G27
Different kinds of specification.

Specification Quality Control **Concept *051**

Specification Quality Control (SQC) is a rigorous specification quality control discipline. SQC is concerned with defect detection, defect measurement, defect removal, process improvement and entry/exit controls. It is based on evaluating specification conformance to specification rules.

Notes:

1. During SQC, specifications are checked against their relevant rules, sources and kin documents for validity. Any rule violations are defects. The density of defects is used to judge the 'quality' of craftsmanship of the specification.
2. SQC includes the Defect Detection Process (DDP) and the Defect Prevention Process (DPP). Both are defined in detail in (Gilb and Graham 1993). When the DPP (process improvement) is used the scope goes beyond quality control and extends to quality assurance.
3. Traditionally, SQC does not pretend to judge the specifications in terms of their relevance or profitability in the real world. It is primarily concerned with making sure that the specifications are clear, complete and consistent by checking a specification and any of its source and kin documents against Specification Rules. It judges whether the specification is suitable to be used in subsequent engineering or management processes. However, by using a different type of rules, Specification Review Rules, it is possible to extend the SQC process to checking the readiness of specifications for review. This could be for a business review or a technical review. See Review *197.

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

Description: Chapter 8, "Specification Quality Control: How to Know How Well You Specified."
Acronym: SQC *051.
Synonyms: Inspection *051; Peer Review *051; "For additional specialized synonyms, see (Wheeler, Brykcznski and Meeson 1996)."
Related Concepts: Quality Control *279; Specification Defect *043; Specification Rule *129; Specification Review Rule *543; Review *197.

SQC — Concept *051

Acronym for 'Specification Quality Control'.

Stakeholder — Concept *233

A stakeholder is any person, group or object, which has some direct or indirect interest in a system. Stakeholders can exercise control over both the immediate system operational characteristics, as well as over long-term system lifecycle considerations (such as portability, lifecycle costs, environmental considerations and decommissioning of the system).
The parameter 'Stakeholder' can be used to specify one or more stakeholders explicitly. We can attach stakeholder information to any elementary specification, or to a set of specifications, as appropriate.

"4.16 Stakeholder: An interested party having a right, share or claim in the system or in its possession of qualities that meet their needs."
Draft Standard ISO/IEC 15288 (ISO/IEC 1999)

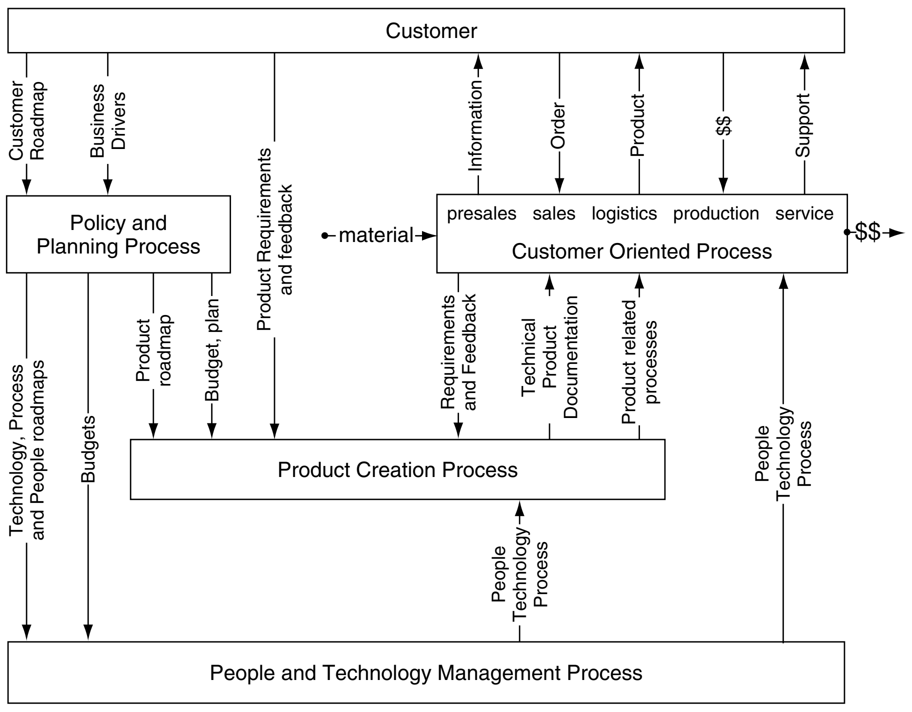

Customer
Customer Roadmap
Business Drivers
Product Requirements and feedback
Information
Order
Product
$$
Support
Policy and Planning Process
material
presales
sales
logistics
production
service
Customer Oriented Process
$$
Technology, Process and People roadmaps
Budgets
Product roadmap
Budget, plan
Requirements and Feedback
Technical Product Documentation
Product related processes
People Technology Process
Product Creation Process
People Technology Process
People and Technology Management Process

Figure G28
Some stakeholder concepts. Courtesy Gerrit Muller, Philips, Eindhoven, NL. (Muller 1999).

//INTEGRA/ELS/PAGINATION/ELSEVIER UK/OMP/3B2/FINALS/0750665076-GLOSSARY.3D – 321 – [321–438/118]
24.6.2005 10:11AM

Notes:

1. The views and needs of stakeholders have to be sought and listened to. For example, stakeholders might have an interest in:
   - setting the requirements for a process, project or product
   - evaluating the quality of a product
   - using the product or system, even indirectly
   - avoiding problems themselves as a result of our product or system
   - the system or product being compatible with another machine or software component
   - determining the constraints on development, operation or retirement of the system or product
2. Stakeholders specify requirements, directly or indirectly, for the system attributes (function, performance, resource, design constraints, and condition constraints). They determine the degree of product or system success or failure.
3. Systems engineers should determine which requirements the stakeholders need, and which requirements they can afford. Even if the stakeholders are not currently conscious of those needs and limitations!

   Example:

   Goal [Stakeholders = {Installers, Service People}, End This Year]: 60 hours <- Marketing Authority.

   Marketing Authority: Stakeholder: Our Service Organization.

   The Goal requirement applies to a set of defined stakeholders. The requirement authority (the one who has requested this Goal level) is defined as another stakeholder.
4. Stakeholders can be internal or external to a system – it depends on the context. Internal stakeholders are typically in our development organization. External stakeholders might be the users and customers of the developed system. Often very external stakeholders are instances like laws and government organizations that can impose requirements on our system. This distinction is useful:
   - to help us develop better lists of stakeholders
   - so we don't get fixated on the 'customer/user' as the only requirements source
   - to give us a systematic set of (internal) stakeholders to deliver to, as we evolve the system, even when it is not ready for external stakeholders.

Related Concepts: Owner *102; Client *235; Sponsor *396; Decision-Maker *237; Consumer *038; User *234; Designer *190.

Standards — Concept *138

A standard is an official, written specification that guides a defined group of people in doing a process. It is a best-known practice.

"A thing serving as a recognized example or principle to which others conform or should conform or by which the accuracy or quality of others is judged."

Oxford Dictionary[3]

[3] *The New Shorter Oxford English Dictionary*, 1993. Oxford: Oxford University Press. ISBN 0-19-861134-X.

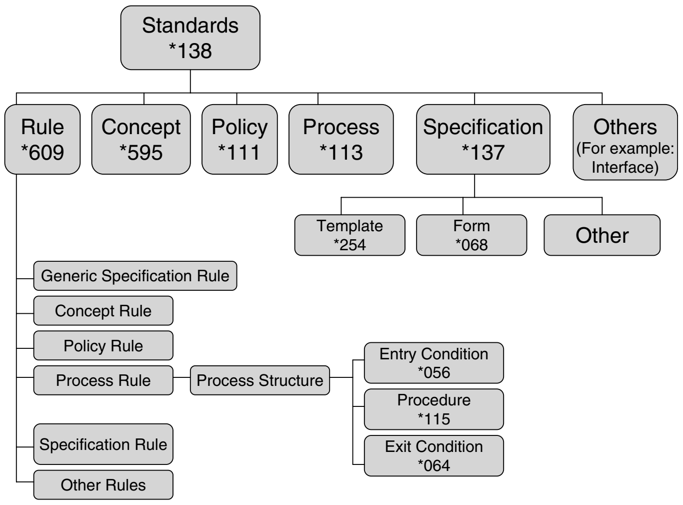


Figure G29
Shows a variety of work process standards provided by Planguage to help define work processes.

Standards include: {rule, policy, process, entry condition, procedure, exit condition, form, template}.
Synonyms: Work Process Standards *138.
Related Concepts: Rule *129; Policy *111; Process *113; Form *068; Template *254.

Status **Concept *174**

Status is the outcome of an evaluation of a defined condition (or set of conditions). Status can be a matter of establishing true/false or it can be a set of different indicators (status settings). Status determines whether a specification or system component applies/is usable or not.
Notes:
1. For a specification, an evaluation of its status is done whenever a qualifier or a conditional statement is evaluated. Status is implied.
   Example:
   Goal [By End of Year, USA, Manager, Product X, If Customer Y Signed]: 500 Items.
   The Goal only applies (500 Items is a Goal) if all the qualifying [time, place and event] conditions are met (Status = true).
2. An explicit Status specification can also be made.
   Example:
   Safe: Status: {A, B, C, Not D, Weekday}.
   A: Condition: Everybody feels good and no one panics.
   B: Event Condition: No official alarm is raised by Building Safety, or on public address.
   C: Indicator Condition: No red light flashing on your workstation to warn of unsafe air conditions.
   D: Sign Condition: Lobby Sign says ‘‘No Smog.’’

Different types of Condition are examined. Status is used to collect the results of the condition set. The reason you might organize things this way is to provide clarity of specification and to enable reuse of specifications. For example 'No Alarms: Status: {B, C}.'

3. Status is commonly explicitly used as a parameter for classifying *specification status.*
   (The underlying conditions are usually not explicitly specified with a specification status.)
   Example:
   Document XYZ: Version: February 22, 2005. Status: Draft.
   This can be used for a document, or any defined specification, including a single requirement or design specification. Suggested status settings (indicators) include:
   - Undetermined
   - Under Revision
   - Exited
   - Approved
   - Validated
   - Verified (proven to be present and correct by some form or test or observation)
   - 'Initial, Defined, Agreed Upon, Released' for 'working state' (as used by Daimler Chrysler, 2002 [Personal Communication 2002]).

   Rationale:
   - to clearly warn specification readers when a specification is not really approved for certain uses.
   - to allow even small subsets of a larger specification document to be independently upgraded or downgraded in status.
   - to help control the evolution of any technical specification.
4. Status can also be used for *system component control.* For example, at the beginning and end of a process, the relevant entry and exit conditions respectively are usually status-checked. Each entry/exit condition could have a true or false status.

Synonyms: State *174.
Related Concepts: Condition *024.

Stretch — Concept *404

A Stretch parameter is used to define a *somewhat more ambitious* target level than the *committed* Goal or Budget levels.

A Stretch level is specified on a defined Scale, under specified conditions [time, place, event]. There is *no commitment* to deliver a Stretch level. Stretch announces that there is *some* stakeholder value at that level, *if* we can find a *practical* or *economic* solution for delivering it.

Notes:

1. The intention is that a Stretch target is challenging, even quite difficult to attain.
   It is used in an attempt to *inspire and motivate* people to do their very best and to do something more than they would *otherwise* dare to do.
2. There is *not a project commitment* to attain a Stretch target. The technology to reach it may be unknown or unavailable. The technology could be too expensive at present to make it profitable to make

this level the Goal or Budget level. However, if without using any extra resources, the project could reach a Stretch level, it would be welcomed. It would have some potential stakeholder value as a result.

3. A Stretch level specification can be a resource target or a performance target.

"Stretch is reaching for more than what you thought possible. . . . In a stretch environment, the same field team is asked to come in with 'operating plans' that reflect their dreams – the highest number they think they had a shot at: their 'stretch'. The discussion revolves around new directions and growth, energizing stuff. . . . We'll never stop 'stretching'."
Jack Welch former CEO General Electric, in Jack: Straight from the Gut *(Welch 2001 Pages 385–6)*

Synonyms: Stretch Target *404; Stretch Level *404 (see Level *337).
Related Concepts: Need *599; Target *048; Goal *109; Budget *480; Wish *244.
Keyed Icon: >+ In context: --->+--->O--->+--->
Historical Note: The Stretch concept was first used in Planguage by Pete Fuenfhausen, then at Nokia, Dallas, TX, September 1999.

Supports — Concept *415

'Supports' is used to indicate what an attribute is *mainly intended to* support. It differs from 'Impacts,' which can include information about all the negative unintended side effects. 'Supports' only lists selected intended main supporting impacts.

Example:
Low RF Power Output [Radio Heads]:
Supports: {Availability, Co-existing, Robustness, Others} <- Marketing Specification 4.2.1.8.

Related Concepts: Impacts *334; Is Supported By *414.

Survival — Concept *440

Survival is a state where the system can exist. Outside the survival range is a 'dead' system caused by a specific attribute level being outside the survival range. For example, 'frozen to death' or 'suffocated.'

A Survival *parameter* specifies the upper or lower acceptable limits under specified conditions [time, place, event], for a scalar attribute. It is a *constraint* notion used to express the attribute levels, which define the survival of the entire system.

For example, a system violating a Survival limit becomes illegal, or totally unprofitable, or in strong violation of a contract. Survival limits are typically derived from laws, regulations and contractual specifications.

Each survival specification should always have a clearly stated Authority or Source specified.

Notes:

1. Survival is used to clearly state the nearby existence of a strong 'sudden death' borderline for an entire system. Worse than a Survival level is a 'catastrophe.'

2. One elementary scalar requirement can have several simultaneous Survival specifications. This is because there can be different qualifiers for each Survival level (that is, different times, places and events can apply). For example, different stakeholders can set different criteria.
3. Survival can be used to set resource limits or performance limits, at both extremes (---[--- and ---]--->O---[--- and ---]--->).
4. Survival implies strong authority behind it (like a Law or Corporate Policy). You should always document the exact source of this authority, using Source and/or Authority. You should also include any other information, which that would help the specification reader to understand why this requirement has been classified as a Survival Limit (such as Rationale).
5. A Survival Limit violation will *not necessarily* lead to *real* catastrophic failure. The failure degree depends on discovery and reaction from the Authority behind it at the time and place of violation. For example, just because the heart stops does not mean the person is finally dead. However, the heart cannot be expected to start up on its own: death may well result.

Example:
Financial Cost Budget:
Scale: Cost in $ for Total Project.
Budget [Lifetime Warranty]: $9.5 million. Rationale: To allow for risks and any lawsuits.
Fail [By Contract Completion]: $9 million. Rationale: To ensure Profit Level.
Survival [By Contract Completion]: $10 million <- Contract 5.4.3.
Budget, Fail and Survival specify requirements with varying priority. Budget implies 'get to this level for success.' Fail specifies 'must reach this to avoid any failure (disappointment in the results).' Survival sets the upper financial limit to avoid disaster.

Synonyms: Survival Level (see Level *337); Survival Limit (see Limit *606).
Related Concepts: Fail *098; Limit *606.
Keyed Icon: [and/or] "The '[' being a lower limit and the ']' being an upper limit."

Systecture © — Concept *564

See Systems Architecture *564. Systecture is a conjunction of the terms 'systems' and 'architecture'.

Historical Note: In July 2002, in connection with a book manuscript on systems architecture, I needed a catchy term for the book title. In my 1988 book, Principles of Software Engineering Management, *I had coined the terms 'softecture' and 'softect'. So, it seemed natural to extend this to the system engineering area. A web search turned up* www.systect.com *(Systect, Inc. 'The system architects') a systems architecture company, but no use of Systecture at all.* © Tom@Gilb.com *2002. Permission is granted to use the term as a generic word. I felt there was a need to get away from the 'architecture' term. Architect is from 'Archi-Tecton,' which means 'Master Builder.' 'Archi' is not from 'Arch', but from 'Arche': primitive, original, primary.*[4]

[4] Contributed by Niels Malotaux.

System [Planguage] ***Concept *145***

A system is any useful subset of the universe that we choose to specify. It can be conceptual or real. In Planguage, a system can be described fundamentally by a set of attributes. The attributes are of the following types:

- function: 'what' the system does
- performance: 'how good' (quality, resource saving, workload capacity)
- resource: 'at what cost' (resource expenditure)
- design: 'by what means.'

In addition, other factors describing various aspects of the system can be specified. These include:

- requirements
- dependencies
- risks
- priorities.

All these specifications (the attributes and the additional factors) are qualified by time, place and event conditions.

Notes:

1. There are specific Planguage parameters for capturing all the system information, including: Function, Performance, Resource, Design, Requirement, Dependency, Risk and Priority.
2. A Norwegian professor client of mine said (about 1969) that he detested the word 'system' because it "had the precision of the word 'thing'." I have ever since then been careful using it, and hope the Planguage definition limits the scope somewhat.
3. Here are some standard definitions of 'system':
   Standard Definition [System, ISO 9000, 2000]:

   "An object consisting of interrelated or interacting elements."

   Note, this ISO 9000 definition emphasizes the internal relationship or interaction of system elements. This has limited interest. The most central aspect of systems is how they are externally experienced and perceived by other systems, so the Planguage definition emphasizes the attributes and admits the possibility of all manner of description, including the 'interacting elements' – but chooses to emphasize real-world 'interaction' (between any one system and all others).
   Standard Definition [System, EIA/IS-731.1, 1996 Interim Standard]:

   "system: The aggregation of end products and enabling products that achieves a given purpose."

   Note in this EIA/IS-731.1 system definition, the concept of 'purpose' comes in. However, lost is the possibility of multiple stakeholders and multiple purposes through time.
   Standard Definition [System, ISO/IEC 15288, preliminary version 2000]:

   "4.17 System An object consisting of interrelated or interacting elements (ISO 9000: 2000).
   NOTE: In practice, a system is 'in the eye of the beholder' and the interpretation of its meaning is frequently clarified by the use of an associative noun, e.g. product system, aircraft system. Alternatively the word system may

be substituted simply by a context dependent synonym, e.g. product, aircraft, though this may then obscure a system principles perspective."

Note in this ISO/IEC 15288 definition, the authors seem to see a problem with the concept, but try to solve it by encouraging specific adjectives to describe it. They stick to the official version [ISO] but do not mention attributes or purposes. A hint about systems principles is given. Standard Definition [MIL-STD 499B]:

"System: An integrated composite of people, products, and processes that provide a capability to satisfy a stated need or objective."

Note this MIL-STD 499B definition is unnecessarily narrow (it does not include the Planetary system, or the molecular system, ☺) and unnecessarily broad (a people or product or process would be sufficiently narrow for many systems engineering purposes). It is good that it mentions the capability to satisfy requirements, but some systems have capabilities that satisfy nobody's requirements (like faults and side effects). Systems are as they are, whether we like it or not. We have to be able to understand and describe their attributes realistically, like them or not.

4. When defining a system, it is important to decide the relationship between the system being observed and changed, and the system of the people (the project) bringing about any change. Numerous different relationships can exist. At one extreme, a project can be completely within the system being modified. At another extreme, a project might be developing a product system to be sold into various, as yet unknown, target systems.

Synonyms: System *145; Object *099: A separate concept number has been allocated as the two terms tend to be used distinctly.
Related Concepts: Attribute *003.

Systems Architecture — Concept *564

Systems Architecture is the set of artifacts produced by Architecture Engineering. A systems architecture is a strategic framework and consists of models, standards and design constraints specifying mandatory and recommended best practice for implementing and maintaining systems.

Notes:

1. A systems architecture usually applies across a division or an entire organization.
2. A systems architecture varies in its level of detail depending on its maturity and what is required of it. Different organizational cultures will require different things. The main point is that a systems architecture should be cost-effective.
3. The aims of a systems architecture could include:
   - imparting technical strategy
   - sharing best practices
   - ensuring specific standards are adhered to (for example, security)
   - avoiding duplication of effort
   - reducing risk by promoting tried and trusted information
   - encouraging recognition and use of standard interfaces
   - promoting reuse

- ensuring compatibility of data structures amongst systems
- achieving economies of scale through standard platforms (especially for training, support and maintenance).

4. Individual systems will have their own architecture (Architecture *192), which will adhere to any relevant mandatory systems architecture.

Synonyms: Systecture *564.

Related Concepts: Architecture Engineering *499; Architecture Specification *617; Standards*138; Architecture *192.

Systems Engineering — Concept *223

Systems Engineering (SE) is an engineering process encompassing and managing all relevant system stakeholders requirements, as well as all design solutions, and necessary technology, economic and political areas.
The fundamental purposes of systems engineering are to:

- optimize the system solution at the highest level of stakeholder concerns,
- synchronize all contributing disciplines to contribute efficiently to the final system characteristics,
- consider the entire system life cycle needs,
- manage risks for the entire system and the entire system life.

An INCOSE Definition:

> "Systems Engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation. Systems Engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs."
>
> *(http://www.incose.org/whatis.html)*

Blanchard's Department of Defence (DoD) Definition:

> Systems Engineering is the "process that shall:
> 1. Transform operational needs and requirements into an integrated system design solution through concurrent consideration of all life-cycle needs (i.e., development, manufacturing, test and evaluation, verification, deployment, operations, support, training and disposal);
> 2. Ensure the compatibility, interoperability, and integration of all functional and physical interfaces and ensure that system definition and design reflect the requirements for all system elements (i.e., hardware, software, facilities, people, data); and
> 3. Characterize and manage technical risks."

(Blanchard 1997)

An FAA (the USA Federal Aviation Authority) Definition:

> *Systems Engineering is:* "A hybrid methodology that combines policy, analysis, design, and management. It ensures that a complex man-made system or product, selected from the range of options available, is the one most likely to satisfy the customer's objectives in the context of long-term future operation or market environments.
> Systems engineering is applied throughout the system or product life cycle as a comprehensive, possibly iterative, interleaved, or recursive, technical process to*:*
> *a.* Translate an operational need into a configured system or product meeting the operational need
> *b.* Integrate the technical contributions of all available development resources, including all technical disciplines into a coordinated effort that meets established program cost, schedule and performance objectives. This involves a 'holistic view' (the design of the whole as

distinguished from the design of the parts). Such a view is multi-disciplinary in nature, rather than disciplinary or interdisciplinary;

c. Ensure the compatibility of all function and physical interfaces (internal and external)

d. Ensure that system or product definition and design reflect the requirements in system or product elements (outcome, hardware, software, facilities, people, and data).

e. Characterize [identify, define, and classify] technical risks, develop risk abatement approaches, and reduce technical risks by prevention and mitigation of impacts when risks are realized."

Source: FAA-iCMM Appraisal Method Version 1.0 A-19, INCOSE Conference CD, June 1999, Brighton UK (FAA 1998).

Notes:

1. The Systems Engineering process is a conscious attempt to avoid sub-optimal engineering. Without Systems Engineering, the success of the resulting system is more accidental than predictable. Systems engineering is necessary because there are so many possible places for product development to go wrong. For example, sub-optimal results might be caused by setting requirements for too narrow a list of stakeholders, or by using too narrow a set of design ideas to solve the problem of satisfying all project requirements. Another frequent problem, especially in well-established large companies, is for groups to produce optimal components yet produce a very sub-optimal complete system.

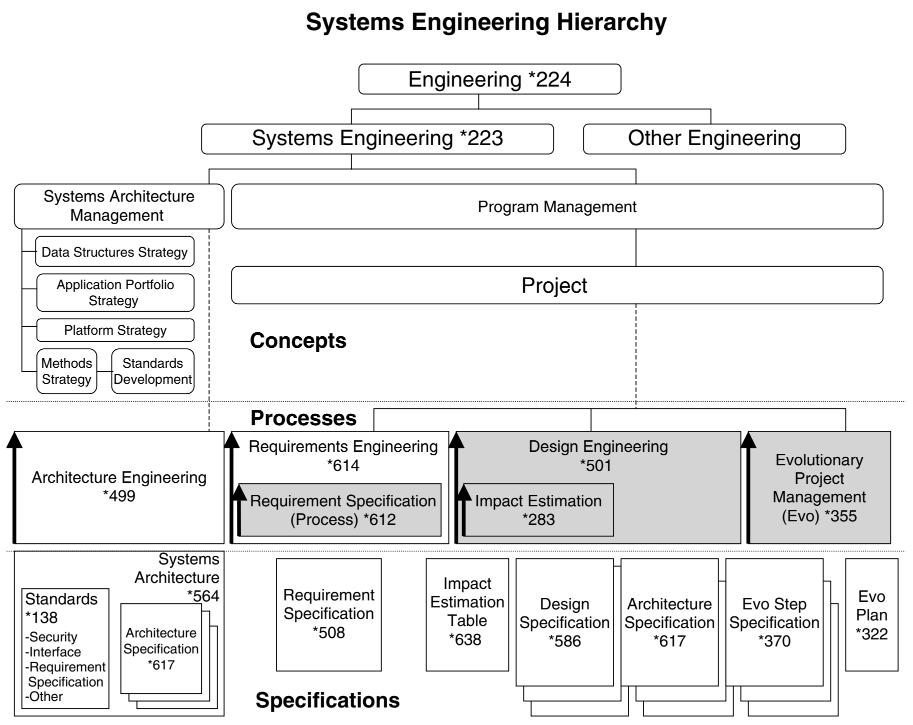

Systems Engineering Hierarchy
Engineering *224
Systems Engineering *223
Other Engineering
Systems Architecture Management
Program Management
Data Structures Strategy
Application Portfolio Strategy
Platform Strategy
Methods Strategy
Standards Development
Project
Concepts
Processes
Architecture Engineering *499
Requirements Engineering *614
Requirement Specification (Process) *612
Design Engineering *501
Impact Estimation *283
Evolutionary Project Management (Evo) *355
Systems Architecture *564
Standards *138
-Security
-Interface
-Requirement Specification
-Other
Architecture Specification *617
Requirement Specification *508
Impact Estimation Table *638
Design Specification *586
Architecture Specification *617
Evo Step Specification *370
Evo Plan *322
Specifications

Figure G30
Shows the relationship for systems engineering amongst concepts, processes and specifications.

2. Systems engineering includes a broad application of disciplines such as requirements engineering, quality control, project management, test engineering and any of the many other disciplines that might be found useful for satisfying stakeholders. Architecture Engineering, a subset of systems engineering, is by contrast, directed only towards the design aspects.

Synonyms: System Engineering *223.
Related Concepts: Engineering *224; Architecture Engineering *499; Requirement Engineering *614; Design Engineering *501; Evolutionary Project Management *355.

Tag **Concept *146**

A term that serves to identify a statement, or set of statements, unambiguously.

Notes:

1. A local tag name is declared for any set of specification statements which needs a 'unique identity' to enable local cross-referencing. It is a direct reference to the specific set of specification statements. For example, 'Local Tag 3.' A local tag must be unique within the specification it is declared in, without needing hierarchical tag references or any other device to locate it.
2. Hierarchical tags help people navigate, and understand the context of a specification. A hierarchical tag identifies a local tag name in some larger context. It can also be used to resolve ambiguity amongst two or more identical, local tags. For example, A.B.Button.XY and C.D. Button.XY. A hierarchical tag has a structure of Tag 1.Tag 2.Tag 3. . . . .Local Tag N. Meaning Tag 3 is a subset of Tag 2. Tag 2 is a subset of Tag 1 – and that Local Tag N will be found in Tag 1's location.
   Hierarchical tags can have any useful number of levels, no matter how many levels are defined in total in a specification. You do not have to repeat the entire formal sequence of hierarchical tags – just specify enough to help find the local tag you are referring to, unambiguously.
3. One use for hierarchical tagging is to identify a specific Planguage statement. For example, Usability.Fail and Usability.Goal refer to the parameters under the Usability tag.
4. [Qualifier] conditions can be used to differentiate amongst several equal terms. The qualifier conditions act as an extension to the normal tag helping us to distinguish amongst different spaces within the scope of the same tag name.

   Example:
   Reliability [USA, Retail Dealers].
   Project Oasis.Requirements.Usability.Fail [USA, Retail Dealers].

Synonyms: Identifier *146; Tag Name *146.

Target **Concept *048**

A target is a specified stakeholder-valued requirement, which you are aiming to deliver under specified conditions. There are two kinds of target: 'scalar' and 'binary.' Scalar targets are specified using the parameters {Goal, Budget, Stretch, Wish}. A performance target is known as a 'goal' and a resource target is known as a 'budget.' Binary targets are function targets.

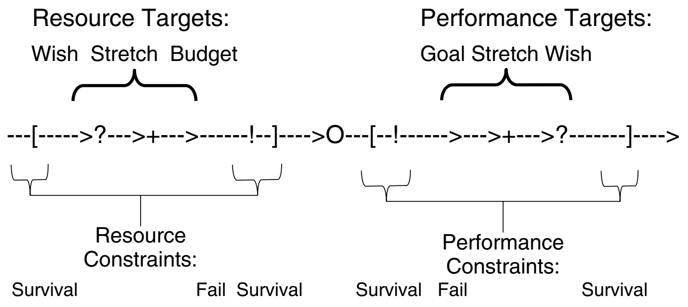


Figure G31
Shows scalar targets can be specified for both performance and resource requirements.

Notes:
1. A target is not a constraint. Targets specify the *success* levels for scalar requirements, and any stakeholder-desired binary requirements. Constraints by contrast specify any *failure* and/or *survival* levels for scalar requirements, and any *mandatory* requirements for binary requirements.
2. Function targets, valued functions, are subtly different from function constraints. Function constraints are mandatory. If a function constraint is not met, then some degree of failure will occur, or even total system catastrophe. Function targets do not have implied penalties, they are considered required by some stakeholder.
3. A scalar target specification consists of a numeric value (its target level) and its relevant [qualifiers]. As well as Goal, Budget, Stretch and Wish, 'Ideal' is a target parameter, but it is rarely used.
4. Target can also be used as a collective noun, applied to a set of function targets and variable scalar targets with their individual qualified levels.

Related Concepts: Goal *439: a performance target; Budget *421: a resource target; Benchmark *007; Constraint *218; Function Target *420.
Keyed Icon: @

Task **Concept *149**

A task is a defined and limited piece of work. A task may be defined formally by a procedure and other standards (how to carry out a task). A process is characterized by the repetition of a task.
Notes:
1. A task may be a complex activity. It can be defined using a set of process standards, such as procedures, rules, forms, rates, best practice models, checklists, and guidelines.

"A piece of assigned work."

<- *The American Heritage Dictionary.*

2. A task is the main part of a defined process. Entry conditions are checked before we invest time carrying out the task. We also check that a task has met our defined exit conditions before we consider ourselves properly done with it. This structure of a process is sometimes abbreviated as 'ETX' (Entry Task Exit).

Related Concepts: Act [PDSA] *172; Process *113; Procedure *115.

Test **Concept *256**

To test is to plan and execute an analytical process on any system, product or process, where we attempt to understand if the system performs as expected, or not.

A Test parameter can be used to reference specific test plans and processes.

Notes:

1. The overall aim of testing is to determine if the requirements are met.
2. Testing is a means of understanding *how* something works without necessarily understanding exactly why it works that way. Testing is from outside the 'black box.' Only by examining actual construction, or specification can we analyze the inner workings of the system (the design and construction).
3. We typically test by putting planned or random inputs into a system and comparing the resulting outputs (behavior and data) with our expectations. When the outputs deviate from expected behavior, we must analyze the reason to see if the system is failing to meet requirements or if the requirements are wrongly specified. If it is economic to do so, we will probably correct either the system itself, or the specification, or both.
4. The term testing *could* be applied in a much wider sense to *any* form of examination (QC, SQC, QA), but it is specifically limited in Planguage to 'input-output' testing of the system prior to operational use. 'Test' is not intended to apply to *conceptual* models of a system, but only to prototypes and real-world systems.
5. The Test parameter can also be used to specify, or more likely cross-reference, already-developed test cases and test plans for reuse or modification.

   Example:
   Requirement X:
   Scale: . . . .
   Meter [Module Level]: . . . , [Customer Acceptance]: Contract Section 6.0, [Operation]: . . .
   TP XYZ: Test Plan [System XYZ]: Test Plan Document XYZ [May This Year].
   Test [Goal]: TP XYZ [Section 1.2.3, Test Cases = {3.4, 6.7, 9.1.4}].
   Test parameter used to cross-reference test plans and test cases for Requirement X.
   Test Tag: Test [Acceptance]: Two independent observers, [Operation]: Built-in software test.
   A Test specification, which can be included with any attribute requirement, can be referred to via its tag ('Test Tag'). Notice that the qualifiers distinguish between two different stages of testing. The two suggested test methods are very roughly specified. This is useful at early stages of specification in order to get some idea and agreement about test costs and quality. This does not prevent a later stage of engineering detailing this to any interesting level of test plan.
6. Compare 'Test' with the parameter Meter, which is the specification of how to measure numerically according to a defined process in the Meter specification.

Related Concepts: Meter *093: Test differs from 'Meter' in that Test is very specifically concerned with system testing, while Meter is concerned with any form of measurement (for example, SQC).

Time **Concept *153**

Time defines '*when*'. It relates to any notion of time: clock-time or timescale. Time is used both as a parameter and in a qualifier condition.

Example:
Goal [Time = Weekends, Place = UK]: 60%.
Fail [Opening Hours, USA]: 50%.
Opening Hours: Time: {Weekday [0800 to 2000], Weekend [0900 to 1800], Not Legal Holiday}.
Goal [Time = By End of This Year]: 40%.
Goal [Date = Before January 31, This Year]: 30%.

Related Concepts: Place *107; Event *062; Qualifier *124; Scope *419.

Trend **Concept *155**

A Trend parameter is used to specify how we expect or estimate attribute levels to be in the future. It is used as a benchmark.

A Trend parameter states a numeric value, on a defined Scale, under specified conditions [time, place, event], for a scalar attribute that is extrapolated into the future, based on current knowledge.

Rationale: The purpose of Trend is to give us a better comparison (benchmark) for the degree of improvement or change we are planning or achieving, than we would get by using the more static benchmark 'Past.' Trend makes us plan to cope with the future, not just the past. It makes systems engineers think about the *competition.*

Example:
Peace:
Scale: Probability of Peacetime Situation.
Trend [Next Year, Country = {Gb, F, NO, DK}]: 80%? <- Marketing Guess.
Peacetime Situation: National Military Forces are not deployed anywhere for any purpose, even NATO or UN peacekeeping.
Trend represents our expectation of what the Past levels of this attribute will extrapolate to in the future (unless we plan to change that projected reality . . .).

> "The other beauty ('truths I've learned to challenge') goes something like this: A team comes in with a proposal to leapfrog the current position of its leading competitor. The implicit assumption is the competition will be sleeping. Doesn't usually happen that way . . . It was tough, but we tried like hell to look at every new product plan in the context of what the smartest competitor could do to trump us. Never underestimate the other guy."
>
> Jack Welch, former CEO General Electric (Welch 2001 Page 391)

Synonyms: Trend Level *155.
Related Concepts: Benchmark *007.
Keyed Icon: ?<
"Symbolizing a Past (<) with some doubt (?) about the perfect truth.
Must normally be applied on a scalar arrow, <------?<-----O----?<---->"
Historical Note: This concept was suggested first by Kai Thomas Gilb, May 1995.

Type **Concept *398**

Type specifies the category of a Planguage concept. Categories for Type may be defined both by Planguage and by local extensions.

Notes:

1. Type classifications can be explicit.

   Example:
   Maintaining Standards:
   Type: Function. "Explicit specification of Type."
2. Type classifications can be implicit. Type can be implied by content and context.

   Example:
   Usability: Objective. "Implicit use of Type."
   Requirements Section. "Implicit Type given by use in a heading."
3. In this glossary, Type can be used explicitly to state the categories of the Planguage concepts (to save space, these have been omitted from this book).
4. Type can be specified as a hierarchy.

   Example:
   Requirements.Performance.Quality.
5. A specific specification type may *demand* certain rules of specification are followed, or *imply* certain properties. For example, a 'Performance' type will always require a defined scale of measure as it is scalar, but 'Design' will not, as it is binary.

Uncertainty **Concept *310**

Uncertainty is the degree to which we are in doubt about how an impact estimate, or measurement, of an attribute reflects reality. We are 'uncertain' as to whether the current or future reality is better or worse (than the observed or estimated value of an attribute), and by how much it differs.

Notes:

1. The reason behind the uncertainty could be either the expected, known variance in the results, or it could be the quality (accuracy, reliability, precision and relevance) of the measuring or estimating method, or both.
2. A 'risk' is a factor that could result in a future negative consequence. An uncertainty becomes a 'risk' when it implies a potential that a *future* result will be *negative* in relation to a planned or estimated target. (I am well aware of the field definitions used in Economics for risk and uncertainty (Bernstein 1996), and the history of making a distinction (for example, the work of Frank Knight and J. M. Keynes). However, the definitions here are tailored to system engineering purposes, rather than Economics.)

Related Concepts: Risk *309.

Until **Concept *551**

'Until' is a logical operator that is used to limit the extent of a scalar range of values. The purpose is to explicitly map a range rather than have it implied by a single value at one extreme (like a Fail limit).

Example:
Fail [Gb, Next Version]: 60% Until Survival.
Survival [International, Next Year]: 20% Until 0%.
Fail [EU]: 80% Until 20%.
Related Concepts: Or Worse *549; Or Better *550; Range *552.

User-Defined Term **Concept *530**

In Planguage, a user-defined term is a definition of a term made by a Planguage user. It is not a Planguage term (like Scale or Goal), nor a customer-tailored Planguage term, such as a new parameter, parameter synonym or grammatical variation.
The scope of a user-defined term is 'local,' and it might apply within a specific definition, within a specific project or across an organization.
A user-defined term has a tag that it is referred to by. It may be specified using 'Defined' or 'Defined As' or, by adding a tag to give a tag name to any expression, statement, or term.

Example:
Address Change: Defined As: A change to an existing address.
MTBF: Scale: Mean Time Between Failure.
PB: Goal [If Peace]: 20%.
User-defined terms are Address Change, MTBF, PB, Failure and Peace. Address Change is an example of explicit definition. MTBF and PB are tags defined in the statements above. Failure and Peace are defined elsewhere.

Notes:
1. A user-defined term is not part of a Project Language (also known as a 'Specific Project Specification Language' – a customized Planguage Specification Language).

Synonyms: Project-Defined Term *530.
Related Concepts: Planguage Term *211; Project Language *247.

Value **Concept *269**

Value is *perceived* benefit: that is, the benefit we think we will get from something.

Notes:
1. Value is the potential *consequence* of system attributes, for one or more stakeholders.
2. Value is not linearly related to a system improvement: for example, a small change in an attribute level could add immense perceived value for one group of stakeholders for relatively low cost.
3. Value is the *perceived* usefulness, worth, utility or importance of a defined system component or system state, for defined stakeholders, under specified conditions.

"One man's meat is another man's poison." *Old proverb*

4. 'Benefit' is when some perceived value is actually *produced by*, a defined system.
5. Value is relative to a stakeholder: it is not absolute. Quality, for example, is stated in terms of the objective level of 'how well' a system performs, irrespective of how this level is appreciated by any stakeholders. Some defined levels of quality only have a value to some

stakeholders. The same is true for all attributes. There are many Planguage ways of indicating that a stakeholder values an attribute. These include using Value, Stakeholder, Authority, Impacts, and Source parameters.

"Nowadays, people know the cost of everything and the value of nothing."
Oscar Wilde.

Synonyms: Worth *269.
Related Concepts: Benefit *009; Impacts *334; Values *592.

Version — Concept *332

A version is an initial or changed specification instance. A version *identifier* can be made from any symbols. It is useful to indicate unique instances of a specification, also probably the *sequence* of changes, and perhaps even the exact *time* of change.
A version identifier is specified by the Version parameter. By default, use the date as the version identifier.
Notes:
1. The version should be specified at the level of individual elementary requirement and design specifications.
   Rationale: This aids change control. It allows reviewers to focus mainly on the changes themselves, rather than the entirety of large documents, which contain perhaps only a few changes. It also enables us to treat individual elementary specifications as relatively independent objects, which are electronically grouped as needed into useful views, rather than the traditional 'documents.'
   Example:
   Version: January 9, 2003.
   Edition: 1.02 [Feb 21 03 3:54:36 pm].
2. If a date alone is specified on the same line as a tag, and immediately after it, then that date will be understood as the version identifier for whatever that tag encompasses.
   Example:
   Usability: January 9, 2003. Scale: Time to <learn>. Goal: 6 hours or better.
   Usability: Version = January 9, 2003. Scale: Time to <learn>. Goal: 6 hours or better.
   Usability [Version = January 9, 2003]: Scale: Time to <learn>. Goal: 6 hours or better.

Synonyms: Edition *332; Instance *332.

Vision — Concept *422

A vision is an idea about a future state, which is very long range and probably idealistic, maybe even unrealistic.
The future state is likely to be about the position of a corporation or product line in relation to the market – rather than about specific properties of a specific product or system.

Example:
"I say to you today, my friends, that in spite of the difficulties and frustrations of the moment I still have a dream. It is a dream deeply rooted in the American dream.

I have a dream that one day this nation will rise up and live out the true meaning of its creed — 'We hold these truths to be self evident, that all men are created equal'."

Martin Luther King Jr., Washington, DC[5]

Notes:

1. Top managers or leaders state a vision in order to create teamwork to move in a required direction. A vision can be analyzed and decomposed into a set of requirements.
2. In the speeches and writings of senior management, the vision might be the only defined component which is quoted. However, the organization and management would be wise to articulate and clarify their understanding of, and commitment to, the vision by decomposing it into a hierarchical set of specific objectives, including specific goals for the elementary objectives. They should map the path to the vision with both short-term and long-term *numeric* targets. They can then begin an evolutionary process of moving towards the specified vision.
3. A vision statement is the reference point for developing more-detailed specifications, such as product line performance specifications, that support the achievement of the vision. A vision statement presumes that at least an assumption is made about the mission, for example that 'we are in the mobile phone business,' or 'we make aircraft'.

Wish **Concept *244**

A Wish parameter is used to specify a stakeholder-valued, uncommitted target level for a scalar attribute. A Wish level is specified on a defined Scale, under specified conditions [time, place, event]. There is no commitment to deliver a Wish level. A Wish parameter simply specifies some stakeholders' desired level, without considering its cost or practicality.

Notes:

1. Wish parameters can be useful for acknowledging and recording stakeholder desires (while clearly *not* committing to them) *until* suitable design ideas are identified, *until* resources are provided for those designs or perhaps *until* deadlines are adjusted.
2. Wish belongs to the set of target specifications: {Goal/Budget, Stretch, Wish}.
3. Subject to qualifying conditions, Wish specifications have the lowest scalar requirement priority. (The order from highest to lowest priority is Survival, Fail, Goal, Stretch and then Wish.)
4. A Wish specification can apply to a performance or resource requirement.

Rationale: If we did not have a Wish parameter to articulate uncommitted stakeholder needs, then this information might never be collected, and maintained. So, we might lose the competitive advantage of knowing what our stakeholders desire and value, when the resources or technology ultimately become available.

[5] From: http://www.ku.edu/carrie/docs/texts/mlkdream.html/. Martin Luther King, 'I Have a Dream' speech on August 28, 1963, Washington, DC.

Synonyms: Wish Target *244; Wish Level *244 (see Level *337).

Related Concepts: Need *599; Target *048; Goal *109; Budget *480; Stretch *404; Ideal *328.

Keyed Icon: >? "A perhaps questionable goal or budget. In context: ---->?--->O--->?---> "

Historical Note: The Wish parameter was first suggested in December 1995 by the Scottish Widows organization, through Dorothy Graham of Grove Consultants.

Workload Capacity **Concept *459**

Workload capacity is a performance attribute. It is used to express the capacity of a system to carry out its workload, that is 'how much' a system can do, did or will do.

Notes:

1. Workload capacity can be used to capture many different concepts of workload, such as maximum number of registered users, maximum number of concurrent users, maximum data volumes and average transaction response times.
2. Workload capacity expresses the system capability to perform a defined type of work.

Scale [Generic]: An amount of defined [Work Task] to be done in a defined [Time] by a defined [Agent] in a defined [Environment] on a defined [System].

Synonyms: Work Capacity *459; Workload Capability *459; Workload *459; Capacity *459.

Related Concepts: Performance *434; Quality *125; Resource Saving *429; Workload Capacity Requirement *544.