Electronic File Headers [Glossary Planguage MASTER]

### TRY OUR DIGITAL VERSION OF THE GLOSSARY AT

<http://concepts.gilb.com/definition-Software-Engineering?structure=Glossary&page_ref_id=534>

Or in our Tool

[ValPlan.net](http://ValPlan.net)

In Mac Pages, you should activate the View Table of Contents, to get a fast index to the glossary main terms. 070619 New. Similar in Word.

### Owner: Tom Gilb

### © 1996-9, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 (risk, OK etc), Innovation, Expert, 2017 Stakeholder, \*683 Implementation, 2019 (Incremental ∆, )

Email comments: Tom@Gilb.com

Web Location: [www.Gilb.com](http://www.Gilb.com), “Competitive Engineering” manuscript.

VERSIONS ON WEBSITE: , free version at [gilb.com](http://gilb.com) from 2019

This contains an abbreviated (10%) glossary and much discussion in whole book o f the concepts

Oct 2014 Full Planguage Concept Glossary

In Downloads/Gilb Essentials

<http://www.gilb.com/dl830>

VIEW THIS WITH INDEX BY

PULL DOWN MENU

**VIEW/SIDEBAR/DOCUMENT MAP PANE (works best in DocX, not Pages)**

You can then find terms alphabetically easily

Oct 2014 Full Planguage Concept Glossary

In Downloads/Gilb Essentials

http://www.gilb.com/dl830

http://concepts.gilb.com/Glossary

GILB.COM SITE GLOSSARY

Plicons: A Graphic Planning Language for Systems Engineering

(Plicons Paper)

http://www.gilb.com/DL37

Planguage Concept Glossary as edited in Competitive Engineering book 2005 (10% of Full Glossary)

http://www.gilb.com/DL387

OLD Full Glossary http://www.gilb.com/DL386

Paper on the Concept Glossary:

A Conceptual Glossary for Systems Engineering: Define the Concept, don’t quibble about the terms.

<http://www.gilb.com/DL46>

THIS GLOSSARY IS THE FULL PLANGUAGE GLOSSARY

( AND IS INDEPENDENT OF THE CE GLOSSARY EDIT)

### Intended Audience: Any Reader Of A ‘Planguage Series’ book (manuscript).

{Competitive Engineering (Manus 1996-2004), Evolutionary Project Management (Manus1997+), Requirements Engineering (Manus 1998+), Priority Management (Draft 2000), Future: {Impact Estimation, Specification Quality Control, Design.}.

Value Planning 2014-2017, BOOKS WRITTEN 2018 (Life Design, Innovative Creativity, Clear Communication, Technoscopes, 100 Practical Project Planning Principles). And in 2019 (Value Requirements, Value Design, Value Management, Sustainability Planning, and Value Agie). Books digitally in www.gilb.com

### Filename: Glossary “Glossary PLANGUAGE MASTER 080317.pages

Administration file

Glossary ADMIN Master.pages

### Homepage: [www.Gilb.com](http://www.Result-Planning.com)

### Quality Control Level: not inspected or reviewed.

**Latest Version**: March 31, 2015

\*307, \*552,

July 3 2017 Stakeholder: added NV Krishna sources

**Interesting Glossary sources:**

http://www.donald-firesmith.com

http://www.seecforum.unisa.edu.au/glossary/definition.asp

### Language Variants:

#### Norwegian Language many terms chosen March 99, separate file (Gilb).

### ========== The above is NOT intended for publication in a book =========

# 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, in ‘Physics and Philosophy’

With the Heisenberg quotation in mind, this glossary will try to give the interpretation Planguage intends, when the 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[[1]](#footnote-1).)

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](http://www.result-Gilbplanning.com).

### 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 concepts are defined, although not used directly in this book, because I have experienced a need amongst my students and clients for me to take a position on the use of them.

Some other concepts have been included because the glossary has been developed in connection with drafting future books in this Planguage 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](http://www.Gilbresult-planning.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 use any terms they wish to use, to point to a concept definition, as needed. 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 it freely, and partially, when suitable credit is given to origin. Notification of your use would always be interesting to me and may result in useful updates and feedback to you (Notification of use, reference, and © credit to [Tom@Gilb.com](mailto:Tom@Gilb@Result-Planning.com) (which is sufficient credit annotation)).

### 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.

# Lead capital letters will be used for glossary terms in this book (most commonly in the glossary) whenever it seems appropriate to remind the reader that there is a formal definition of the word.

# THE GLOSSARY ALPHABETICALLY

## Abbreviation Concept \*454 August 6 2001

A Planguage parameter meaning ‘abbreviation’.

Type: Planguage parameter.

Synonym: Abbr.

## Absolute Specification Concept \*509 March 4, 2003 C

An absolute specification of a term can be understood fully without reference to other definitions.

Notes:

1. It would be for example a number (3.14), a number, a date (Jan 1), a standard code (‘NY’ or ‘GB’), or a proper name (‘San Francisco’), rather than a Planguage term, a Project Language term, or a user-defined term.

Related Concepts [Absolute Specification \*509]:

• Planguage Term \*211: A term defined by Planguage.

• Project Language \*247: A Planguage-variant term defined by a project.

• User-Defined Term \*530

Type [Absolute Specification \*509]: Specification.

## Abstract Strategy Concept \*660 August 13 2008 (Born)

An abstract strategy is an idea or state, without *itself* containing or being a Concrete Strategy (\*661).

Notes:

In particular all types of • Strategic Objectives (\*214) and Means Objectives (\*215) are abstract strategies,

Objectives (\*100) and its synonym Performance Requirements (\*100) are normally abstract strategies unless they do not pretend to support any higher level objectives – which would be the case with Core Values and Fundamental Values (See book Built To Last) and other Fundamental Objectives.

A Concrete Strategy, by contrast is a real world function with multiple attributes which impact higher level objectives. The Abstract strategy is merely a single dimensional attribute (like higher reliability) with a given level of that attribute, and without any concrete strategy. Any useful concrete strategies may now or later be appointed to serve the purpose of delivering the result or an abstract strategy.

Related Concepts [Abstract Strategy \*660]: <- Vision Engineering paper Aug 13 2008

• Concrete Strategy (\*661)

• Strategic Objectives (\*214)

• Means Objectives (\*215)

• Strategy (\*047)

• Performance Requirement (\*100)

• Objective (\*100)

Type [Abstract Strategy \*660]: Specification.

## Acceptable Range \*545 May 29, 2003

This is a performance or cost range that the stakeholder will tolerate, or find acceptable. This does not mean that they are happy or that the system is successful.

•••••••••••••[!!!!!!!!!!======>\_\_\_\_\_\_>O•••••••••••••[!!!!!!!!!!======>\_\_\_\_\_\_>

•••• is a catastrophe range. [ is a lower survival level. !!!!! is a Failure Range.

==== is an acceptable range. >is a Goal level. \_\_\_\_\_ is a success range.

Notes:

1. Partial payment can be an agreed consequence of being in this range.
2. An acceptable range is a scalar range which is neither a Success Range, a Failure Range, or a Catastrophe Range.
3. It extends from just below a Goal level and until any constraint level, like Fail, is reached, if any is defined.
4. It can also be viewed as extending from below the Goal level to and including the Tolerable level.
5. There can be more than one acceptable range for each unique set of qualifier conditions.

Synonyms [Acceptable Range, \*545]

• Tolerable Range \*545

Related Concepts [Acceptable Range, \*545]

• Range \*55

• Failure Range \*546 not an acceptable range

• Catastrophe Range \*603 not an acceptable range

• Success Range \*548 better than just ‘acceptable’ range

• Acceptable Level \*613 any numeric level in the acceptable range.

Type [Acceptable Range, \*545]: scalar concept.

## Acceptable Level \*613 March 3, 2003

Any numeric level in the acceptable range.

The acceptable range extends from just below a Goal level and until a constraint level is reached, if any.

This is a ‘not success’ and ‘not failure or catastrophe’ level. Tolerable.

## Acquisition Concept \*356 March 8, 2003

Acquisition is the activity of obtaining a component. Acquisition is part of the Development Cycle.

Notes:

1. The intended use of ‘acquisition’ in a systems engineering context, is the acquiring of system components. The components can be anything intended to help reach results. Acquisition can include any process necessary to reach the required system state for a component such as {bidding, request for proposals, negotiation, contracting, paying, financing, testing, getting approvals, quality control, building, maintenance}.

For example: acquired components can include: {equipment, permissions, tools, methods, advice, expertise}.

Examples of use of term in context: ‘architecture acquisition’, ‘step acquisition’, ‘step acquisition costs’.

Related Concepts [Acquisition \*356]:

• Development Cycle \*413

Type [Acquisition \*356]: Process.

## Act [PDSA] Concept \*172 June 4, 2003

Act [PDSA] is part of the Plan Do Study Act Cycle. 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).

*“to standardize, conclude and theorize”*

W. E. Deming [Personal Communication 1991]

Notes:

1. 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.

Related Concepts [Act [PDSA] \*172]:

• Plan Do Study Act Cycle \*168

• Plan [PDSA] \*169

• Do [PDSA] \*170

• Study [PDSA] \*171

• Evolutionary Project Management \*355

• Statistical Process Control (SPC) \*466

Keyed Icon [Act [PDSA] \*172]: There is no keyed icon for Act [PDSA].

Drawn Icon [Act [PDSA] \*172]: Act is the south side of the rectangular process symbol.

The four corners of the process drawn icon stand for the Shewhart process-cycle definition of ‘Plan, Do, Study, Act’.

Type [Act [PDSA] \*172]: Process.

## Act. (Noun) Concept \*278. January 23, 2002

An ‘act’ is a process of doing something, by means of humans or other Agents.

*Note: This definition is to help us distinguish between the PDSA ‘Act’ defined above (\*172) and this conventional use of the word.*

*Related Concepts Act (noun) \*278:*

*Not to be confused with*

*• Act \*278 (part of PDSA cycle)*

• Process \*113. Process is the set of Entry Process, Task and Exit process, and is formally defined or definable. All formal processes contain tasks. A task alone is not a formal defined process.

• Task "task" : \*149 . A task is the work done when executing a defined procedure

*Synonym: action*

*Type: general systems concept*

## Action: Concept \*538. March 30 2002

Action is a specification parameter which describes an action to be taken under defined circumstances.

Type: Activity specification

Keyed Icon [Action, \*538] ^[<action specification or tag>]. Same as Process.

Example:

Minimum [EU, Children] 15 degrees C, Action [Close School].

\_ [EU, Children] 15 degrees C, ^ [Close School].

Parameter: Action

Related Concepts:

• Process \*113

• Procedure \*115

\*Act \*278

• Act \*172

## Actual Step Content. Concept \*369 January 3, 2003

The design ideas, modified by [actual qualifiers] and functionality modified by [actual qualifiers], which are actually delivered in a specific Evo step.

Rationale: we needed a concept to distinguish between plans and reality.

Note: The planned step content is detailed in the step specification. The actual step content must be observed, logged and measured.

Related Concept [Step Content \*369]: Step \*141.

Type: analytical tool

## Adaptive (adjective). Concept \*563. July 2, 2002

‘Adaptive’ refers to the set of concepts which are interesting in relation to changing a system from one defined state to another. This can be while a system is in operation (user changes operational parameters). It can refer to system modification such as porting, extending, improving, correcting.

## Add, Concept \*643 ‘-’ November 18, 2003

Replace is a Planguage verb indicating the addition of some specification to a defined set of specifications.

Keyed Icon [Add \*643] ‘+‘ in context referring to a defined set of specs.

Related Concepts [Remove \*642]

• Replace \*641

• Remove \*641

Type: Planguage Grammar

## After Concept \*313 February 12, 2003

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

Example:

Product Trials: Step: {F2, F1 After F2, F1 => F3}.

This means that Product Trials (a tag) is a defined Evo ‘Step consisting of step elements F2, and then F1, and then F3 in that sequence.

Keyed Icon [After \*313] => and <=

The 'precedence 'arrow'

A => B means A before B and B after A

Type [After \*313]: Parameter [Logical].

## After Concept \*313 February 6, 2003

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

Example:

Product Trials: Step: {F2, F1 After F2, F1 => F3}.

This means that Product Trials (a tag) is a defined Evo Step consisting of step elements F2, and then F1, and then F3 in that sequence.

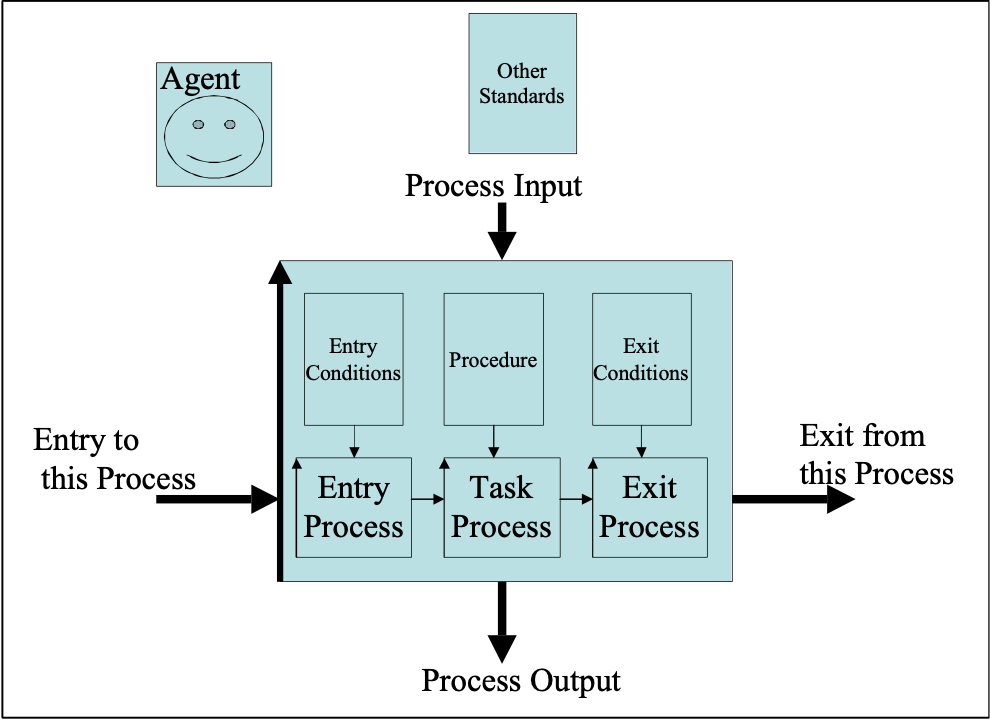
Keyed Icon [After \*313]: =>

*“*The same icon is used for *both* ‘Before’ and ‘After*.’* Only the positions of the related tags need be positioned correctly, <‘*Before’ event tag*> => <‘*After’ event tag*>.”

Type [After \*313]: Logical Operator.

## Agent Concept \*163 May 23, 2003

Any combination of person, group, and/or machine, which carries out a defined (written) procedure, thus doing a task, which is a part of a process (other parts being entry and exit sub-processes.

*Figure \*163: An agent carries out a task, according to a defined procedure, which applies defined standards. This makes up the heart of a process, which consumes process input, and produces the process output.*

Type [Agent \*163]: Systems Engineering Concept.

## Agile (Adjective) Concept \* 677 18 Oct 2014

“Any set of tactics that enable a prioritized stream of useful results, in spite of a changing environment”

Initially defined by T Gilb TsG 7 June 2013 for Bank Client Glasgow

## Aim Concept \*001 February 3, 2003

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)

“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.” <-[DEMING93, page 52]

*“The aim must include plans for the future” <- [DEMING93, page 51]*

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>.”

“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?”

<- [DEMING93, page 52] Attributed to Deming by Carolyn Bailey

Example:

Aim [New System X]: Superior long term competitive edge in all market areas and product lines.

Keyed Icon [Aim \*001]: @ “The same icon as Target.”

In context, O--@-->

The distinction between the ‘Aim’ and ‘Target’ concepts is the level of system and detail. ‘Aim’ is generally a higher overall concept than goal or target.

Related Concepts [Aim \*001]:

• Goal \*109

• Budget \*480

• Target \*048

• Mission \*097

• Vision \*422

Type [Aim \*001]: Requirement Type, Parameter.

## Alternative Designs: Concept \*588

*Alternative* designs will have satisfactory, but *different*, performance and cost attributes. The alternative design idea attributes will be so significant that they do not need any *other* alternative design ideas in addition, or cannot afford it

Related Concepts [Alternative Designs, \*588]

• Supplementary Designs \*589

## Ambition (Level): \*423 February 17, 2003, 3Feb2015 ‘Level’, 210618 User Story

‘Ambition’ is a parameter, which can be used to summarize the ambition *level* of a performance or budget 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.

3. [210621] A User Story (Mike Cohn blog.mountaingoatsoftware.com, and our Paper [User Stories Need Reinforcements

http://www.gilb.com/DL461

www.agilerecord.com/agilerecord\_06.pdf]

A user story might be directly used as an Ambition level statement, especially when it has a clear indication of a 'level' of ambition, rather than being a function or a design. Sometimes a design idea (I need a password) implies a quality of value level (high security), and this might be included in an edited User Story. (I as a user need a password for high security)

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.

Serviceability:

Ambition: The PEXX System shall be extremely easy to learn to service and to service for a variety of service people.

Accessibility:

Ambition: The PEXX System shall contain and support any forms of access, which are prioritized by our stakeholders.

Security:

Ambition: Security levels for protection of the system in operation, and under a variety of threats (intrusion, corruption, and destruction) shall be sufficiently high to protect our customers and our vital interests. They shall also be at such a level that they are a ‘killer argument’ for using our products in general.

Adaptability:

Ambition: The system extension, modification and porting to other environments will be relatively easy and inexpensive by design of the architecture.

Examples of real disguised ambition levels drafts. These are not to be confused with the detailed specifications that follow them with Scale etc. They are not final specifications; they are the beginning of a specification process; or a summary of the specified detail.

Example:

Simplicity [PEXX System]:

Ambition: The system must be as simple as possible for any user to learn to use, and to use.

Impacts: {<defined strategic requirements>, for example Machine Utilization }.

Impacted by: {<a set of Designs which impact this Requirement>, for example ‘Look&Feel’, Multi-Lingual}.

Type: Quality Requirement.

Stakeholders: {Operator, Supervisor, Instructor, Manual Writer, Production Planner, Salesperson, …}.

Scale: The number of distinct work Operations a defined [User] must do, in order to accomplish a defined [Task]

Test: Test sets for defined [Operations] using defined [Users], are observed electronically and measured and compared.

========== Benchmarks ======= Analytical background ======

Past [FCXX II, User = Novice Operator, Task = Status Inquiry]: 23±5 <-Estimate by Jan.

Record: 0.

Trend: ?

=========== Targets === real requirements =========

Wish [<Qualify when, where, if>]: 0. “On the defined Scale, always”

Goal [FXXXX + PEXX System, User = Novice Operator, Task = Status Inquiry, Release 2.0, If TopaXX is installed at this site]: 10 Operations, [Task = Change Date]: 5.

Stretch: 4.

=========== Constraints ========================

Fail: 5.

============= Other possible options =============

Risk: The Operator is not trained properly.

Assumption: The User Manual is accessible and updated and will be used.

Basis: No known basis for this is identified <-TG.

Design: < some info about expected designs but NOT a required one>.

Owner: Jan V.

Version: 0.1

This is an example of an Ambition statement, together with a **full** real draft requirement. It is not perfect, but it is real, with some XX to retain confidentiality.

Keyed Icon [Ambition \*003]: @.∑ “Target and Summary.”

Type [Ambition \*003]: Parameter [Scalar].

Related Concepts: Vision \*422, Story \*665

## And Concept \*045 February 5, 2003

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

Examples:

Goal [If War and Inflation]: 60%.

Goal [If Peace And Inflation]: 60%.

Goal [If War AND Stability]: 60%.

Goal [If War & Peace]: 60%.

To make a statement read better, the lead capital letter can be dropped, giving ‘and’ rather than ‘And’.

Keyed Icon [And \*045]: &

Type [And \*045]: Logical Operator.

## Architecture (collective noun): Concept \*192. May 9 2005, Jan 6 2013

The ‘architecture’ is the set of entities that

•  in fact exist and impact a set of system attributes directly,

• or indirectly, by

• constraining, or influencing, related engineering decisions.

Notes:

The ‘architecture’ is the set of perceivable design components, that deliver the observable system attributes. <- added Jan 2012

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 who, if anyone, consciously specified the architecture design artifacts.

• Inherited Architecture: architecture which was not consciously selected 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 Maier02, including a special appendix on the history of attempts to define a standard in DoD, IEEE, INCOSE. Appendix C pp283-9. In addition the book gives a great many other insights into the nature of the concept.

Note: XE "Architecture [Noun]." \*202.Architecture [noun] XE "\*202.Architecture [noun]"

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 thesecomponents?)· relationships between components (How do components relate inspace and time?)· dynamic interplay description (How is control passed between andamong components?)· flows (How does data or in-process product flow in space andtime?)· resources (What resources are consumed where, in the process orsystem?).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:** D **(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*

## Architecture Engineering Concept \*499 May 29, 2003

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 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.

The architecture engineering process 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.

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 [Architecture Engineering \*499]:

• Architecting \*499

Related Concepts [Architecture Engineering \*499]:

• Systems Architecture \*564

• Requirement Engineering \*614

• Design Engineering \*501

• Architecture Specification \*617

Type [Architecture Engineering \*499]: Process.

## Architectural Description [IEEE] Concept \*618 July 18, 2003

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 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 [Architectural Description \*618]:

• 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 (collective noun) \*192: this is the real set of artifacts that the architectural description describes.

Type [Architectural Description \*618]: Specification.

## Architecture Specification Concept \*617 June 17, 2003

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. Architecture Description: This can be broader than a specification, and include models, prototypes, and real systems to aid in describing a real or projected architecture.

(Note: 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.’)

Synonyms [Architecture Specification \*617]:

• Architectural Specification \*617

Related Concepts [Architecture Specification \*617]:

• Architecture (collective noun) \*192

• Architecture Engineering \*499

• Systems Architecture \*564

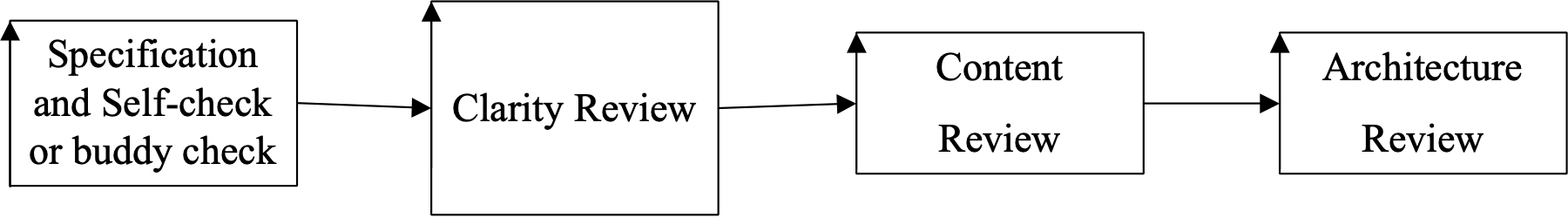
• Architecture Description \*618

Type [Architectural Specification \*617]: Specification.

## Architecture Review: Concept \*596 August 15, 2002

An architecture review is a review process which looks at the *entire* set of engineering artifacts for a given system at a given level of process or development.

It is not limited to the design architecture, but may choose to look at requirements, the architecture of a sub-system, the software architecture, the test planning. But the point is that it looks at a complete set of specified ideas which apply to a defined system.

Fig. \*596 The Architecture review is one type of content review. It can choose many review focus ideas, for example ‘End Life Disposal’, for a variety of purposes.

Related Concepts [Architecture Review, \*596]

• Specification Quality Control, \*051 this just checks rules for clear specification. Is the requirement or design clear, complete, consistent and has necessary background information?

• Specification Concept Review \*542 this just checks that a single artifact is specified according to applicable Content Rules \*543. Is it a good design or good requirement?

Spec \*137 -> Clarity Review \*607 -> Content Review \*542 -> AR \*596 -> next process

## Artifact [Development]: Concept \* "\*227.artifact" 583 January 8, 2004

Development artifacts are engineering information and tools such as specifications, test cases, user documentation which are not regarded as product components, but can potentially be reused to develop products and product components.

Thanks Stefan Ferber, Bosch, for suggesting this term, July 2002.

Synonym:

• engineering concept

• engineering artifact

• development concept

Related Concepts [Artifact [Planguage] "artifact" \*227]:

• entity \*645

• object \*099

• artifact [Planguage ] \*227

## Artifact [Planguage] "artifact" : Concept \*227 "\*227.artifact" . January 8, 2004

A collective term for all Planguage "Planguage" {books, slides, translations, variants, dialects, and other related things}.

Implied use: a Planguage artifact.

Related Concepts [Artifact [Planguage] "artifact" \*227]:

• entity \*645

• object \*099

• artifact [development] \*583

Synonym: Planguage Artifact

## Assertion: Concept \*597. July 24, 2002

An assertion is a statement that something is true, without necessarily backing it up with evidence or proof.

Related Concepts [Assertion, \*597]

• Impact Assertion: a design template parameter that asserts the expected impact of a design without requiring or expecting proof of the assertion.

## Asset Concept \*581 July 15, 2003C

An asset is a *potential* resource.

An asset is anything that can *potentially* be used by a project or a system as a source of resources. Assets are not necessarily being used, or planned used, as resources at all, but they are *potentially* available, if needed.

An asset for a stakeholder is not necessarily owned by that stakeholder.

Notes:

1. Some assets can give rise to resources.

Examples:

NON-REUSABLE/CONSUMABLE ASSETS WITH THEIR RESPECTIVE RESOURCES

ASSET RESOURCE A UNIT OF MEASURE FOR BOTH

(RESOURCE ATTRIBUTE) ASSET AND RESOURCE

TIME Elapsed Time Hours

WAREHOUSE STOCK Raw Materials Quantity/Day

STAFF Effort Work Hours

OFFICES Room Availability Square Meters/Day

Financial Credit/Loan/Low Interest Money

Credibility

2. Assets can be owned and upgraded.

Examples:

'REUSABLE'/UPDATABLE ASSETS:

• DATAFILES/DATABASES: Input Data

• STANDARDS DOCUMENTATION: Standards

• STAFF: Knowledge

• OFFICES: Room Space

3. Processes can modify assets: they can create assets, and they can upgrade assets.

4. Assets can also get transformed by a process into some other thing: an upgraded asset or better performance, improved functionality, etc. For example, sell knowledge (an asset) and get financial resource.

Related Concepts [Asset \*581]:

• Resource \*199

Type [Asset \*581]: Design.

## Assets [Product Line] Concept \*581 July 13, 2002

Any general product line components or development artifacts (specs, test cases etc.) that can potentially be reused for a product are 'assets'.

COMMENT FROM STEFAN:

developing assets usually cost more money than developing just component for single use.

Therefore, you need a concept for the strategic and planned reuse.

We call an asset that is developed and planned for reuse a CORE ASSET.

Thanks to Stefan Ferber, Bosch, for suggesting this concept. July 2002.

## Assignable Causes "Assignable Causes" : Concept \*020. "\*020.Assignable Causes"

Causes, of change in Attributes of a Process, which have a *Common* root Cause.

*Term coined by Dr. Walter Shewhart. "Shewhart." Preferred synonym: common causes <-Deming "common causes" .*

July 9, 2001

## Attached Specifications. Concept \*517. April 20, 2002

Attached specifications, are directly and visibly present specification elements to one given specification.

They are not inherited or implied.

Related Concepts:

• Local \*376. Local specifications are only valid in one specification. An attached specification might, by contrast, refer to other specifications and determine or influence them (inherited specifications).

• inherited specification attributes , \*516. These are not directly or locally attached specifications, which nevertheless have the effect of being attached, because of global effect, or specified effect.

Synonyms: attached specification element, attached sub-specification.

## Attribute Concept \*003 May 12, 2004

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 attributes:

• 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, 1990.

attribute

inherent property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means

[based on ISO/IEC 15939:2002]

Note: ISO 9000 makes the distinction between: inherent means existing in something, especially as a permanent characteristic, and a characteristic assigned to a product, process or system (e.g. the price of a product, the owner of a product) is not a quality characteristic of that product, process or system.

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 [Attribute \*003]:

• Characteristic \*003

• Property \*003

Related Concepts [Attribute \*003]:

• Performance \*434

• Function \*069

• Resource \*199

• Design \*047

Keyed Icons [Attribute \*003]:

There is no specific icon for ‘attribute.’ The icons for the main categories of attribute are used instead, as follows:

• For a function attribute: O “The drawn icon is an oval.”

• For a performance attribute: O-> “An outward pointing arrow from a function.”

• For a resource attribute: ->O “An inwards pointing arrow to a function.”

(The Past, Record, Trend, Fail, Goal, Stretch and Wish scalar values are various points along these arrows and each have their own icon, see the relevant Glossary entries.)

• For a design attribute: [*<design name>*] “The tag name of the design, underlined and enclosed in square brackets.”

Type [Attribute \*003]: System Component.

## Asset Concept \*581 July 15, 2003B

An asset is a *potential* resource.

An asset is anything that can *potentially* be used by a project or a system as a source of resources. Assets are not necessarily being used, or planned used, as resources at all, but they are *potentially* available, but not necessarily ‘owned’, if needed.

Notes:

1. Some assets can give rise to resources.

Examples:

NON-REUSABLE/CONSUMABLE ASSETS WITH THEIR RESPECTIVE RESOURCES

ASSET RESOURCE A UNIT OF MEASURE FOR BOTH

(RESOURCE ATTRIBUTE) ASSET AND RESOURCE

-----------------------------===========================------------------------------------------------------

TIME Elapsed Time Hours

WAREHOUSE STOCK Raw Materials Quantity/Day

STAFF Effort Work Hours

OFFICES Room Availability Square Meters/Day

Financial Credit/Loan/Low Interest Money

Credibility

2. Assets can be owned and upgraded.

Examples:

'REUSABLE'/UPDATABLE ASSETS:

• DATAFILES/DATABASES: Input Data

• STANDARDS DOCUMENTATION: Standards

• STAFF: Knowledge

• OFFICES: Room Space

3. Processes can modify assets: they can create assets, and they can upgrade assets.

4. Assets can also get transformed by a process into some other thing: an upgraded asset or better performance, improved functionality, etc. For example, sell knowledge (an asset) and get financial resource.

Related Concepts [Asset \*581]:

• Resource \*199

Type [Asset \*581]: Design.

## Assets [Product Line] Concept \*581 July 13, 2002

Any general product line components or development artifacts (specs, test cases etc.) that can potentially be reused for a product are 'assets'.

Thanks to Stefan Ferber, Bosch, for suggesting this concept. July 2002.

## Assumption Concept \*002 January 8, 2003

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. donm@softed.com

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:

Assumption: Cell phones will not be available in remote places. Therefore people will buy satellite phones. <- Katherine.

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.

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 be good practice to specify the things that *determine* if this assumption is going to be true. Use Depends On, Authority, Source, Impacted By and similar parameters just below the Assumption statement. See above example.

Related Concepts [Assumption \*002]:

• Basis \*006

• Rationale \*259

• Condition \*024

• Qualifier \*124

• Risk \*309

Type: Parameter, Risk analysis concept.

## Attack Concept \*672 August 14 2014

Attacks are Threats activated.

They attempt to cause negative consequences in a system.

They might be prevented from doing damage by built-in Mitigation.

They might be compensated for, or recovered from, by Post Damage Mitigation



Figure \*672 A: the sequence of threats, some becoming real attacks, confronting built in mitigation, which sometimes fails to deal with the attack, causing damage.

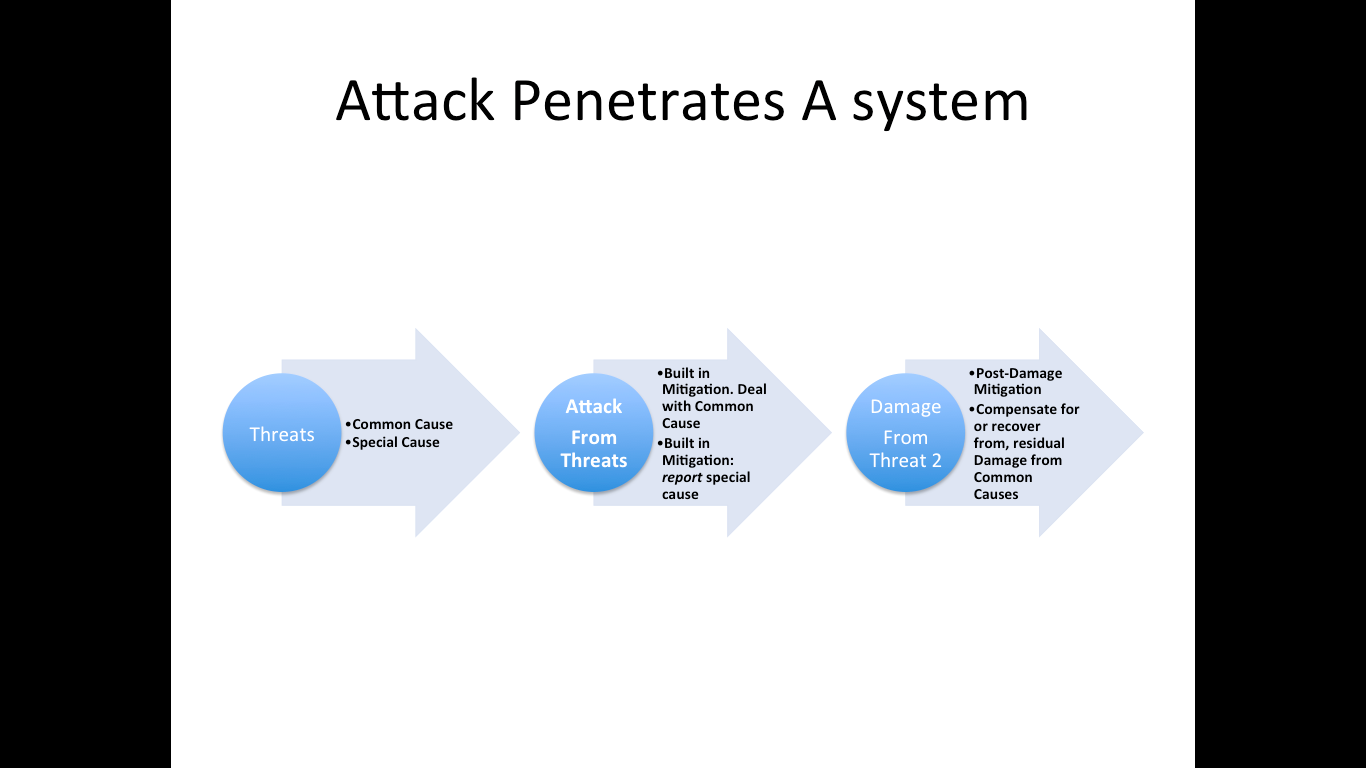


Figure \*672 B: Threats become attacks and try to penetrate a system, with resulting damage; which might be mitigated in the system itself, or post damage (recovery, insurance, compensation, toleration)

Notes:

….

**Synonyms:**

Threat Penetration

**Related Concepts [ Attack \*672]:**

Mitigation \*670

Risk \*309

Threat \*671

Damage \*673

**Type**: Parameter

Attribute Level Icon Point: (Concept \*252) July 9, 2001

This is the exact point in space, in a diagram, which represents a given attribute level.

Related Concept: Generic Attribute Level Icon: (Concept \*251)

Type: graphic icon concept

*Notes:*

* *It is the generic attribute level icon (--|--), and/or the tip of the outermost arrow, in target and benchmark icons.*
* *If desired, the generic attribute level icon can be used, in addition to arrow tips (->|), to make this point more intuitively clear to a reader of the diagram.*

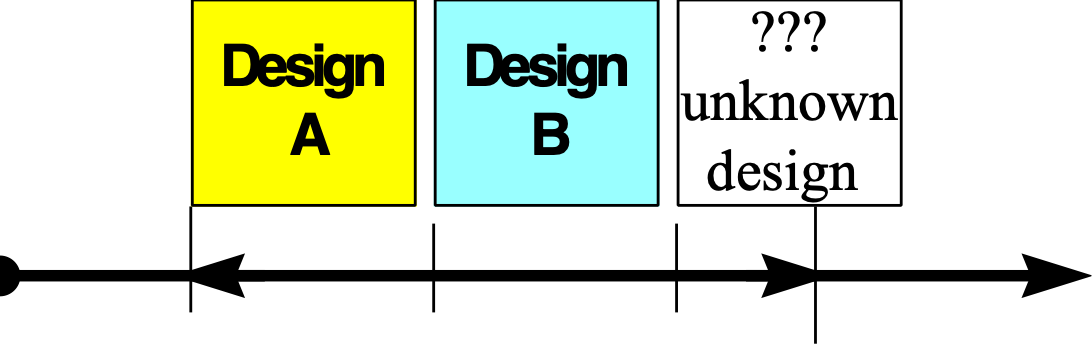
*Example:*

O---- |<-- >>|-------->

The Past level (<-) and the Fail limit (->>) are embellished with the Generic Attribute Level Icon (--|--), in order to emphasize the idea of the exact level, graphically.

*3. The icon can also be used to represent an attribute level reached by a design idea.*

*Example:*



*Figure:*

*Example of use of generic attribute icons. The left-most ‘|’ icon is the symbolic exact ‘0% of impact’ Past point, our chosen Benchmark. The arrow symbol is itself a bit vague as to borders, even though Planguage specifies the tip of the arrow as the precise border. The ’|’ icon between Design A and Design B symbolizes the extent of impact of Design A on this attribute. The next ‘|’ is the extent of Design B impact. The final ‘|’ is the exact extent of a Goal requirement (our chosen Target), ‘100%’ in Impact Estimation terms. It helps symbolize that an ‘unknown design’ might contribute more than the Goal level requirement for this attribute.*

## Author Concept \*004 Nov 12, 2003

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

Note:

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 [Author \*004]:

• Writer \*004

• Specification Writer \*004

Related Concepts [Author \*004]:

• Owner \*102

Type [Author \*004]: Role.

## Authority Concept \*005 January 8, 2003

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 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.

Note: 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].

Type: Parameter, Risk Analysis, Priority.

## Background Concept \*507 May 26, 2003 C

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

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 requirement targets (Goal 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 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.

Examples:

Background parameters include:

• Benchmarks {Past, Record, Trend}

• Owner

• Version

• Stakeholders

• Gist

• Ambition

Related Concepts [Background \*507]:

• Non-Commentary \*294

• Core Specification \*294

• Commentary \*632

• Specification \*137

Type [Background \*507]: Specification.

## Backroom Concept \*342 June 4, 2003

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. Evo project management needs to manage the backroom and frontroom as one synchronized process.

Related Concepts [Backroom \*342]:

• Frontroom \*343

Type [Backroom \*342]: Adjective, Noun.

## Baseline Concept \*351 January 19, 2003 B

A *Baseline* is a stable well-defined benchmark *set* (one or more system attributes) that serve as a comparison for system change.

A *system* baseline is any *set* of system attribute specifications that formally define the state of a system, under specified conditions.

An *attribute* baseline is a *single* *benchmark* that has been chosen to *compare* any system change against (estimated or actual).

Example [System Baseline]

EU Base: Baseline [Europe, 2003]

{Performance Attributes [EU, 2003], Functions [EU, 2003]}.

This baseline is specified without reference to costs.

Note: 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.

Examples:

**QX**:

Type: Quality.

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

Baseline [Our Competitor’s Product]:

Past [Task = Learn to Drive Off, Person Type = Experienced Driver]: 1 minute.

Target [Our New Product]:

Goal [Task = Learn to Drive Off, Person Type = Experienced Driver]: 10 sec.

*This example shows setting an attribute baseline and a target for a quality, QX.*

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.*

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 from the baseline (1 minute) towards the target (10 seconds)).

Design ‘CDI’ would be even worse than the baseline; that is the quality level would be worse than the attribute baseline.

Keyed Icons:

[Baseline \*351] {<<<} “set of benchmarks”.

[Scalar Attribute Baseline]: 0%

Note: A baseline is a benchmark, or set of benchmarks. But if referring to a scalar attribute baseline, then ‘0%’ is more explicit.

However, for ‘[Baseline to Target]\*200]’ pair, the keyed icon is ‘<->’.”

[System Baseline \*145 + \*351] {<<<} [System XYZ]: ….

Meaning ‘baseline for a defined system’. Using the qualifier to define ‘which’ system.

[Benchmark \*007] <<< (related icon).

Related Concept [Baseline \*351]:

• Benchmark \*007

Type [Baseline \*351]: Parameter [Scalar, Relationship], Benchmark

Domain [Baseline \*351]: Impact Estimation, Evolutionary Project Management.

## Baseline to Target Pair Concept \*220 February 11, 2003

In Impact Estimation, this is a set of two levels (two numeric values) associated with a specific requirement under stated conditions: the Scale value for the benchmark reference point (0%) and the Scale value for the target (100%).

Notes:

1. For example, a baseline to target pair, ‘30<->66’ gives the numeric values for the real Scale benchmark (on the left hand side of the icon) and the target (on the right hand side). These values are used to compute the percentage numbers.

2. A baseline to target pair is usually used, without qualifiers in an IE table, as a form of abbreviated documentation of a requirement’s levels. However, the qualifiers in the requirement specification are actually implied, and if necessary should be explicitly stated for unambiguous documentation. The point is that the 100% impact value is only obtained when the target level is reached on time, and with regard to all the other specified conditions.

Keyed Icon [Baseline to Target Pair \*220]: <->

Type [Baseline to Target Pair \*200]: Metric.

## Basic Planguage " Planguage Generics" : Concept \*225 May 19, 2003

Basic Planguage is the original specification of Planguage, as in this book or as updated and expanded on our website, [www.Gilb.com](http://www.GilbResult-Planning.com). It is a version of English (USA spelling format) Planguage material ready for translations to variants and subsets.

*Notes:*

*1. ‘Planguage’ itself varies depending on local variations based on ‘Basic Planguage.’*

2. There is, as yet, no complete concept of standard or standardized Basic Planguage. So users should state the defined version of Basic Planguage that they are referencing. For example, Basic Planguage [Web version, 3.1, January 2005]! The updated website copy is the master version.

*Synonym: Planguage Generics \*225*

*Type: Planguage classification.*

## Basis "BASIS" Concept \*006 January 9, 2003

A basis is an underlying idea that is a foundation for a specification.

A ‘Basis’ parameter is used to specify a foundation idea, so that it is stated explicitly, and 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.

Synonym: Base \*006.

Type: Parameter, risk analysis.

## Before Concept \*312 February 5, 2003

‘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} => 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. The use of the => icon or the term ‘Before’ is a matter of taste; and both are used here for illustrative purposes.

Keyed Icon [Before \*312]: =>

*Note: The same icon is used for both Before and After.* Only the positions of the related tags need be positioned correctly, <’Before’ event Tag> => <’After’ event Tag>.

Type [Before \*312]: Logical Operator.

## Benchmark Concept \*007 January 23rd 2008

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}.

3. It is arguable that the concepts Ideal, Perfect and Wish are also reference benchmarks, rather than actually required states. The Wish is a stakeholder need, but not a formal system requirement. Perfect and Ideal are two classes of extreme states, not likely to be seriously affordable, feasible or prioritized into constraints or targets. If they are in any case really ‘required’ we would need to state them as targets or constraints (added Jan 23 08 tg)

Example:

Usability:

Ambition: Order of magnitude better than future competitors.

Scale: Average time needed to learn to do Typical Tasks for Typical User.

Perfect: 0 seconds.

Estimate [ This Year, Our Old Product] 10 minutes. <- CTO

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’ , ‘Perfect’ and ‘Estimate; are benchmark specifications.

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

Related Concepts [Benchmark \*007]:

• Baseline \*351

• Past \*106

• Record \*127

• Trend \*155

• Estimate \*058

• Perfect \*648

• Ideal \*328

• Wish \*244

Type [Benchmark \*007]: Specification Type.

## Benchmark Cost: Concept \*505. March 16, 2003

A benchmark cost is an expenditure of resource that has been determined by some means for purposes of comparison and analysis.

Benchmark costs are specified using the benchmark level parameters Past, Record and Trend.

Benchmark costs are determined by observation, measurement, estimation, guessing, asking, analysis, approximation, testing, or any other cost-effective means for our purposes.

Synonym:

• Observed Cost \*505

• Cost Benchmark \*505

Type: …..

## Benefit Concept \*009 June 4, 2003

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 [Benefit \*009]:

• Gain \*009

• Profit: Informal use.

* Advantage: Informal use
* Improvement \*009.

Related Concepts [Benefit \*009]:

• Value \*269

• Performance to Cost Ratio \*010

• Value to Cost Ratio \*635

• Effectiveness \*053

• Stakeholder \*233

Type [Benefit \*009]: Systems Concept, Priority Data.

## *Binary Concept \*249 May 10, 2003*

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 [Binary \*249]:

• Scalar \*198

Type [Binary \*249]: Adjective.

## Binary Constraint. Concept \*467 October 8, 2001 tg February 9, 2003

A Binary Constraint is a constraint that only has two possible status conditions: complied with or not. In systems: the specified constraint is either complied with or not. There are two types known as Veto (positive constraint, do X) and Demand constraints (negative constraint, ‘do not do X’).

Type: Constraint Requirement

Related Concepts:

• Constraint \*218

• Scalar Constraint \*468

• Veto Constraint \*462

• Demand Constraint \*461

• Design Constraint \*181

Keyed icon [Binary Constraint \*467] [Constraint Tag] (very similar to qualifier)

Note 1. All qualifiers act as binary constraints, at least on the specification, if not the system or product itself.

## Binding: Concept \*584 July 13, 2002

A formal engineering decision type which constrains variability of product line artifacts or components in order to improve the potential for reuse and commonality.

Thanks Stefan Ferber for suggesting this term, July 2002.

## Black Box "black box" : Concept \*011. August 31, 2001 tg

A Black Box is a conceptual interface to any system of perhaps unknown, or unknowable, internal complexity and/or construction.

*A black box can be tested, or understood, by observing its external behavior. In particular, any system can be judged as to whether its requirements have been met, without necessarily understanding how they were met (i. e. which design ideas were used).*

## *Budget, To Concept \*488 December 20, 2002*

‘To budget’ refers to a process to plan the allocation of limited resources to a system.

The budgeting process results in the specification of a set of target and constraint specifications for one or more resources.

Related Concept [Budget, To \*488]: Budget \*480.

Type: Verb.

## Budget Concept \*480 March 24, 2003 tg

A ‘budget’ is 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 resource target specification).

Example:

Maintenance Effort:

Scale: Total annual Maintenance Engineering Hours per thousand lines of software code supported.

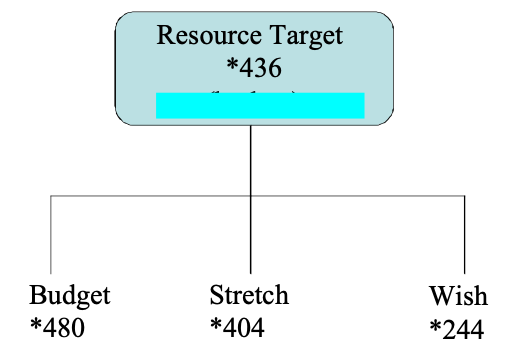
Budget [First Four Years Average]: 10 hours.

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.

Synonyms [budget (the concept) \*480]:

• Budget Level \*480: See Level \*337

• Planned Budget \*480

• Plan [Resource] \*480: Historic usage only

• Planned Level [Resource] \*480: Historic usage only

Related Concepts [budget (the concept) \*480]:

• Aim \*001

• Resource Target \*436

• Target \*048

• Stretch \*404

• Wish \*244

• Ideal \*328

• Goal \*109

• Budget \*480 (the parameter).

Keyed Icon [Budget parameter \*480]: >

“A single arrowhead pointing towards the future. The same icon as for Goal \*109.”

In context: --->--->O

Always use an input arrow to a function oval to represent a resource attribute. 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.

Type [Budget/budget \*480]: a Parameter [Scalar], (‘budget’ the concept) Target.

## Budget Constraint Concept \*421 May 15 2005

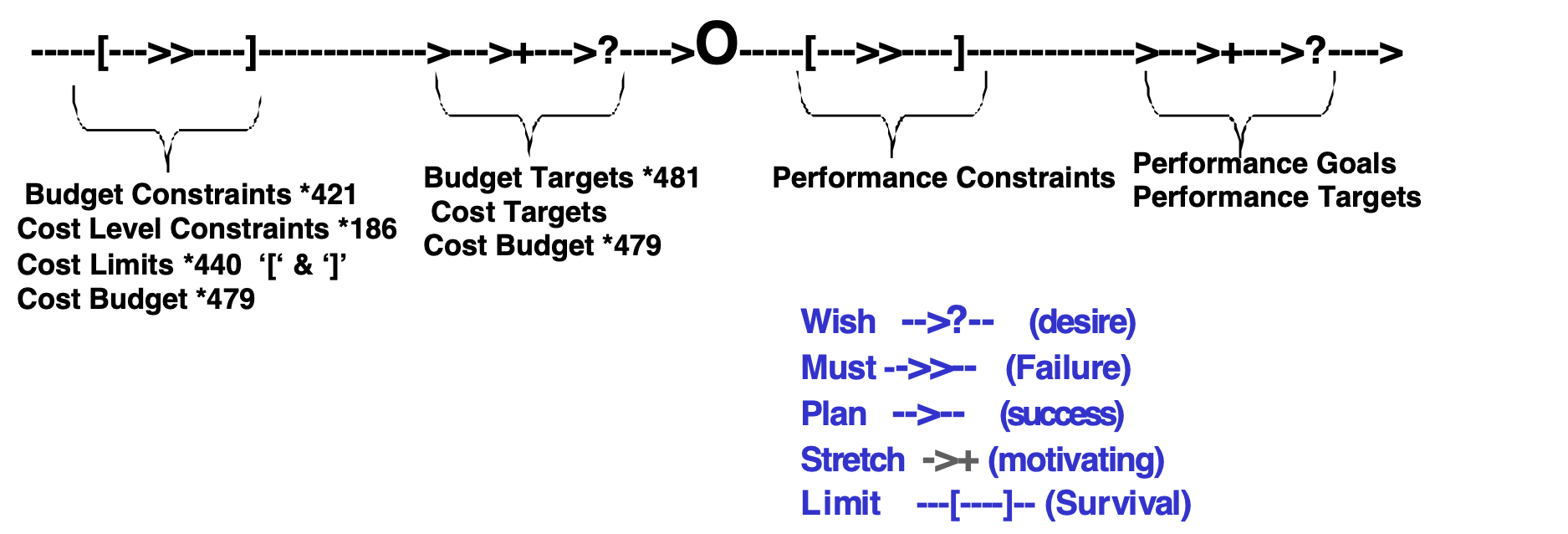
<this was found missing, and is messy in other terms , check all \*421> tg

## Budget Cycle Constraint "budget cycle constraint" : Concept \*012. July 21, 2001

A restriction regarding budget, which is related to the budget cycle process, such as when, who, how much.

## Budget Target: Concept \*481. December 19, 2002

A Budget Target is a numeric resource-expenditure allocation plan. They are expressed with target levels {Budget, Wish, Stretch} .



**NOTE DIAGRAM NEEDS UPDATE FOR NEW terms APRIL 16 02 TG and Aug 6 2002**

Type: resource requirement.

Keyed icon [Budget Targets, \*481]: --->--->+--->?---->O

Related Concepts [Budget Target, \*481]:

• Target \*048, a scalar level we are aiming at more or less.

• Budget \*480, a resource level requirement concept including budget targets and budget constraints

• Performance Target \*439

• Stretch Budget \*404 & \*480, a type of budget target that is more ambitions than the Planned Budget.

• Wish Budget \*244 & \*480. A type of Budget Target which has been noted as a ‘stakeholder desire’ or need, but is not committed (changed to a Planned Budget) with respect to achievement of all other requirements.

• Planned Budget \*109 & \*480. A type of Budget Target which is agreed, formal, committed and hopefully realistic with regard to all committed requirements.

• Budget Constraint \*421, the Fail, Crash and Survival limits are used to specify budget constraints.

Synonyms [Budget Target \*481]:

• Resource Target \*481

## Bug Concept \*339 November 13, 2002

A ‘bug’ is a synonym for a real system fault, which if ‘provoked’ by an agent, or by particular data, can lead a real system to malfunction.

**Error/Slip** 🡪 **Issue**🡪 **Defect**🡪 **Fault/Bug**🡪 **Malfunction.**

The series above is the chain of events leading to malfunction.

Synonym: Fault.

## Business Rules: Concept \* 532. January 14, 2003

Business Rules are specifications or practices that define or constrain some aspect or principle of how a business operates or should operate, it’s ‘rules of behavior’.

Related Concepts:

Specification Rules: (\*609) these are system rules which concern only the best recommended practices for systems engineering specification.

• System Rules, \*531, the generic concept containing the sub-class business rules and specification rules.

• Business Rule Practices. Actual practices whether they are documented or not.

• Business Rule Specifications. Written formal specifications, analytical or planned, whether they are practiced or not.

Keyed Icon [System Rule \*531, and \*523, \*129] : [ ] ->

Example Template: <rule tag>: [Rule Conditions> -> <rule actions>.

Example: Define Scale: [Quality Requirement] -> {Define Scale, Define Meter, Define Targets}.

Business Rules:

This usage is widespread under the name "business rules". This is documented in the work of the Business Rules Group (<http://www.businessrulesgroup.org/brghome.htm>). They say, "A business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business. The business rules which concern the project are atomic ~ that is, they cannot be broken down further" -- except that they can be further broken down into the type of "rule elements" that Don Mills calls CAUSES and EFFECTS.

They continue, "For our purposes, this may be viewed from two perspectives. From the business ... perspective, it pertains to any of the constraints that apply to the behavior of people in the enterprise, from restrictions on smoking to procedures for filling out a purchase order” (specification rules, \*129). “From the information system ... perspective, it pertains to the facts which are recorded as data and constraints on changes to the values of those facts. That is, the concern is what data may or may not be recorded in the information system."

Source [Business Rules]<- Don Mills, NZ Feb 24 2002. Don Mills <donm@softed.com>

## Case "case" : Concept \*213. *July 21, 2001*

Cases are real or doctored-for-discretion (originally real) examples of practice of Planguage (this method).

*They are of any size from a few lines example to a full-blown complete project documentation and background story. They are not necessarily as extensive as full case studies. They are just real instances used to illustrate the theory.*

## Catastrophe Concept \*602 May 20, 2005

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 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 [Catastrophe \*602]: ‘.’ “A full stop or the end on a scalar arrow icon (--->)

Example:

A series of ‘.’ indicates a Catastrophe range \*603, for example:

------->--!--------]……….>O………[-----!-->----------->

The survival limit 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 \*602]:

• Catastrophe Level \*602

• Catastrophe Limit \*602

• Catastrophic Failure: Informal use only

• Death: Informal use only

• Non-Survival: Informal use only

• Tolerable Limit \*602. ‘The start of the intolerable range’ <- Kai Gilb [May 20 2005].

Related Concepts [Catastrophe \*602]:

• Survival \*440

• Range \*552: Catastrophe Range

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

Type [Catastrophe \*602]: Parameter, Scalar level.

## Catastrophe Range Concept \*603 March 3, 2003

The catastrophe range is the full numeric extent of all catastrophe levels.

It can be described by a numeric point where the range begins (Catastrophe range start) because it is assumed to continue as long as the numeric level gets ‘worse’..

Related Concepts [Catastrophe Range, \*603]

• Range \*552 a defined, tagged, extent on a scale of measure

• Catastrophe Level \*602 a numeric value in a catastrophe range

• Catastrophe Range Start \*604 the exact numeric point where a Catastrophe range begins.

Synonyms [Catastrophe Range \*603]:

• catastrophic range \*603

• disaster level/range \*603

• total/absolute failure range \*603

Keyed Icon [Catastrophe Range \*603]

A series of 2 or more ‘•’ or ‘.' ‘•••••••’ or ‘..’ usually in context of an attribute scale (‘••••----->’) and possibly in connection with a survival limit icon (square bracket) for emphasis (‘••••[---->----]•••>O’)

Type [Catastrophe Range \*603]: <Specification>.

## Cell Concept \*308 November 19, 2002

In an Impact Estimation table, a cell is an intersection between a requirement attribute (usually a performance or a resource/cost attribute) and a design idea.

A cell contains information about the specific impact of a design idea on a specific requirement. This information includes an impact estimate, and related data such as the uncertainty, evidence, source and credibility.

Keyed Icon \*308: [] ([ and ] keyed immediately after one another)

Type: Impact Estimation concept.

Domain: Impact Estimation.

## Chance Cause "Chance Cause" : Concept \*019.Chance Cause "\*019.Chance Cause" . July 21, 2001

Synonym of Special Cause "Special Cause” (\*019).

## Change Requirement Concept \*243 March 25, 2003

A ‘change requirement’ represents a ‘differential’ change with reference to a defined ‘baseline’ system.

Notes:

1. Change requirements can involve any type of requirements: Function requirements, performance requirements, resource requirements, and any design or condition constraints. These are the requirements that determine the necessary changes to move a defined system to a new end state.

2. In general, change requirements are intended to represent ‘improvements.’

Keyed Icon [Change Requirement \*243]: +.@ (*Incremental.Target*)

Related Concepts [Change Requirement \*243]:

• Problem \*270

• Need \*599

• Baseline To Target Pair \*220

• Gap \*359: Only for scalar improvements

• Design Target \*338

Type [Change Requirement \*243]: Requirement concept.

## Changing \*631 February 16, 2003

Predefined State. The state is in continuous but predictable flux, not stable.

Related To:

• State \*174

## Check, To [SQC] Concept \*013 May 29, 2003

To ‘check’ a specification means to determine if it is consistent with any applicable specification rules (standards), which it *should* be consistent with.

Note: We check primarily against the applicable rules for a specification. The rules might refer to other documents, or to other parts within this same specific specification, which need to be used to check our specification. The rules might optionally be supported by directly rule-related checklists, which should help ensure the rules are interpreted correctly, and specifically, for the specification type.

Related Concepts [Check, To \*013]:

• SQC \*051

• Checker \*014

• Checking [SQC] \*287

• Specification Rules \*609

• Checklist \*016

Type [Check, To \*013]: Verb.

## Checker Concept \*014 November 20, 2002

A checker is a person, who performs checking within Specification Quality Control (SQC). It is a defined role in the process, and the checking activity can occur in most of the SQC sub-processes, not just the predominant ‘checking’ process itself.

Checkers check specifications against rules, according to defined checking processes, looking mainly for major defects. They will also note and later report, process improvement suggestions; and report ‘questions of intent’ to the author/writer.

Related Concepts:

• SQC (\*051)

• Check (\*013)

• Checking [SQC] (\*287)

• Specification Rule \*609

Type: Role.

Domain: SQC.

## Checking [SQC] Concept \*287 November 20, 2002

Checking is the name of a sub-process within Specification Quality Control (SQC).

‘Checking’ is also the name of a widespread SQC activity, (‘checking’ with a small ‘c’), which although *predominantl*y carried out within the Checking [SQC] sub-process, is also used to some degree within all the other SQC sub-processes. Carrying out ‘checking’ is a core activity in SQC.

Notes:

1. In practice within SQC, checking consists of checking a selected specification against its source and kin documents according formal written rules of specification and consistency. These rules can be supported by checklists. The rules will also indicate which specifications need to be used to check our specific specification (example: sources, kin and other parts of our main specification). The rules used must be those either specifically agreed for production of the specification; or the standard applicable rules for this organization. The checklists used will be those needed to effectively support specific individual rules. The source documents used will be those agreed as relevant inputs for the production of the specificationn. It is the responsibility of a team leader to attempt to determine which rules, checklists, source and kin documents apply, and which shall be utilized during checking. The individual checker can supplement this plan to improve their effectiveness in finding major defects. The degree of checking planned is determined by the cost and effectiveness that is appropriate to the objectives of a particular SQC instance.

2. For peak effectiveness in finding defects, checking must be carried out at an optimum-checking rate (optimum for people’s mental speed of correlating different rules and specifications).

3. The main purpose of checking is to help the document author/writer with useful advice, and to gather data to measure the degree of conformance with specification standards (rules, defect levels specified in entry and exit conditions). There are many and varied purposes depending on the situation.

4. During the Checking [SQC] sub-process, for each cycle of checking, individual checkers typically spend up to two hours; primarily checking for major defects. Identifying potential process improvements, and noting questions of intent to the author/writer, are other activities carried out in larger checking processes.. In ‘sampling checking’, where the primary purpose is to estimate major defect density, as little as half a page might be checked and a range of 5 to 30 minutes might be used per checker. This would be sufficient to determine if the exit level was probably exceeded or not.

Related Concepts [Checking \*287]

• Check \*013

• Checker \*014

• Checking rate \*015

Type: Process.

Domain: SQC.

## Checking Rate Concept \*015 November 20, 2002

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}.

Note: 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

Type: Metric.

Domain: SQC.

## Checklist Concept \*016 June 4, 2003

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.

Example:

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.

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.

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.

4. Note (03 Mar 2018) This is not identical to the checklist concept used for aviation, which is a way of remembering correct content and sequence of a process. That is closer to what we call Rules here.

Type [Checklist \*016]: Standard.

## Chunk Concept \*299 November 20, 2002

A ‘chunk’ is a portion of a document being evaluated by SQC, where the *intent* is to continue checking other chunks after the first chunk is checked, unless the defect data from a chunk is extreme (‘too many’ or ‘zero’ defects).

Notes:

1. A checking sample may be so large that we chunk it in order to handle it in smaller units of work. This is to allow us to ‘sample the sample’ before committing either to more work, or to a specific checking plan or strategy.

2. The major reason for ‘chunking’ is that, when a realistic optimum rate of checking is applied (say 300 non-background words per checker hour), people can only handle smaller parts of a larger document in a reasonable length of time. They may get through one to six physical full pages in say 2 hours. If they try to do an entire document in this time, they may end up using checking rates of 20 pages or more per hour. This speed will reduce their defect-finding effectiveness to less than 5% and so will be a waste of time. It is much more useful to do fewer pages, achieve a higher defect-finding effectiveness (60% to 88% range) and get a deeper insight into the true defect state of the document.

3. In some cases a chunk may turn out to have functioned as a sample. For example, if a two-page chunk of a twenty-page document is checked and 20 Major defects are discovered on each page, then the first ‘chunk’ has served as a sample, to tell us that the rest of the chunks are not worth doing. (Explanation: The defect level is at least an order of magnitude worse than the quality necessary before releasing the document. There is little point in repeating this process for nine more chunks, that is, for the remainder of the document. If we assume 50% defect-finding effectiveness, the defects found represent only about half of the ones that are actually in the checked document pages The entire document must be rewritten to a much higher standard of craftsmanship.)

Type: Artifact.

Domain: SQC.

## Claim: Concept \*553. May 28 2002

‘Claim’ is an assertion by a defined source, about a system attribute or an event. The Claim parameter is used to emphasize that the credibility of the claim is not high, and the reader should exercise their judgement and caution, because there is a risk that the claim might not be correct.

A scalar claim might be so vague that it is non-numeric, and is put in fuzzy brackets.

Example

Implementation Ease:

Scale: the relative speed to implement on a defined set of [Telecoms

Architecture].

Claim: <”appears to be relatively easy”> <- MJ, Paper on DISTRIBUTED FEATURE COMPOSITION: A VIRTUAL ARCHITECTURE FOR TELECOMMUNICATIONS SERVICES

Type: parameter

Domain: description of any attribute or plan.

Keyed Icon: Claim \*553: ? (positioned first on a line as a parameter)

Example

? [USA, This Year]: The sky will fall <- Chicken Little.

Usage:

‘Claim’ should be used when we need to specify that some information is more of an assertion than a well-established fact. It is a signal about a risk. It can also be the first stage of developing better data and evidence, which improves the quality of the data. This could lead to upgrading the specification to another parameter (Past, Record, Trend, and others).

Claim can be used as a prefix to any other parameter, in order to emphasize the risk-filled nature of the information.

Example:

Claim: Past [UK, Last Year] 66%±20%? <- TG oral comment without evidence.

Related Concepts [Claim, \*533]

• Assumption \*002

• Evidence \*063

• Source \*135

• Credibility \*035

Synonym: Assertion \*553.

## Clarity Concept \*400 November 21, 2002

The specification quality of ‘being understood by a defined readership as intended by the writer’.

Notes:

1. We assume that there is a way to express an idea so that a defined readership can understand the intended idea perfectly. That means that clarity is viewed primarily as a property of the specification itself.

2. In system terms, clarity will actually be a function of the author group and their training, motivation and tools, as well as a function of the readership group and its training and tools; and finally, clarity is a function of the specification itself and its resistance to corruption and alteration in the environment it resides in.

3. In addition to the generic rule, “The specification must be clear enough to the intended readership to convey the writer’s intended message”, I have found it useful to add, “The specification must be clear enough that it is obvious how we might test for its existence”.

Type: quality attribute.

Domain: SQC.

## Clarity Review Concept \*607 November 21, 2002

A Clarity Review is a specification review which is primarily used to make sure the specification is clear enough to be interpreted correctly by the intended spec readership, as intended by the spec writers.

In particular, a clarity review is not initially concerned with the engineering goodness of the spec. This is left to later Specification Content Reviews. Clarity Reviews are trying to establish that a specification can be trusted because it is clear, unambiguous, correct, and complete. If so, the basis is laid for deciding if it is good engineering, good architecture and ‘good for business’ – with other analytical processes, which include Specification Content Reviews, experiments, prototypes, Evolutionary Step Delivery, competitive bidding and any other processes which determine how powerful an idea is.

A Clarity Review can be carried out using the generic Specification Quality Control process (SQC). The Rules we check against will tune into these issues. But a clarity review can be less formal – something like a friend checking quickly and declaring it to be fine for them.

Synonym \*607, Clarity Review

• Specification Intelligibility Review \*607 (SIR)

Related Concepts [\*607, Clarity review]

• Spec QC \*051

• Spec Content Review \*542

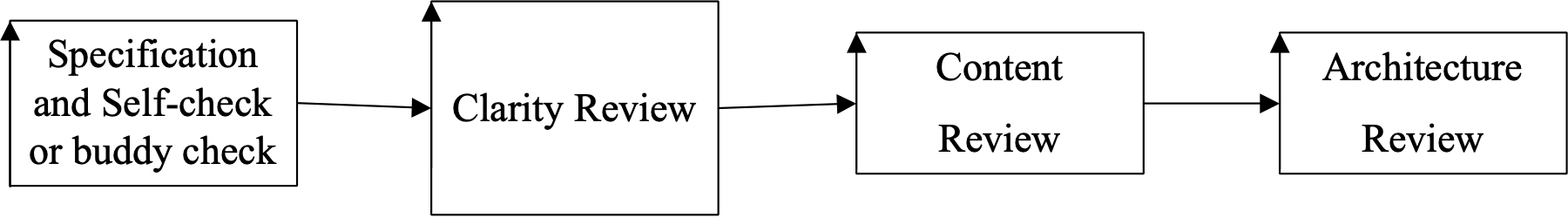


Fig. \*067 The clarity review is the class of specification reviews which has a team check against official rules about clear, complete, consistent, unambiguous specification for intended readership range.

Type: process

Domain: SQC.

## Class [Qualifier] "class" : Concept \*017. March 31, 2015

A class is a defined qualifier term category.

*Note: Some classes are predefined in Planguage like Task, Date, Time, Place, some are defined locally in a particular project. Some classes are used informally and intended to be interpreted as best we can.*

*Goal [Date=25 Jan2001, Place=GB] 60% “formal pre-defined classes”*

*Fail [25 Jan 2001, GB] 55% “implicit definition, ‘guess what class’ based on content”*

*Goal [division =Washers, market segment =private homes] 55% “informal definition” (no caps)*

*Synonyms: Term Class, classification, qualifier class, qualifier term class.*

THE PLANGUAGE CLASSIFICATION PRINCIPLE:

Any thing we specify may usefully be classified in one or more of a wide variety of ways, depending on how and when it is applied. Specification ideas do not have one single classification category exclusively. This applies to all concepts in the Planguage glossary, and concepts not in the glossary. Classifications are useful because we can then access certain rules, ideas, properties and principles that apply to all things in that class. But, things are what they really are, no matter how we classify them!

For example:

A watch:

Can be viewed as a function ('Function = small time telling device): which has certain performance and cost characteristics.

The same watch, can be a design idea for improving a system of process (Use a watch rather than guessing at the time). This will give it the characteristics of watches generally ( a range of attributes which all watches have), or it will give more-specific attributes when a particular watch is chosen ( a design refinement).

The same watch could be a condition:

Goal [ If Watch] ±1 minute accuracy.

## Client Concept \*235 May 6, 2003 TG

A client is a person or group who has requested some defined work, system, or product, and will pay for it, directly or indirectly.

Notes:

1. Use the term ‘Client’ for clarity on this concept. The synonym term ‘Customer’ has lost some of its usual meaning, since it has been used to describe other non-paying people within an organizational chain of work processes (See Consumer \*038).

Synonyms [Client \*235]:

• Customer \*235

Related Concepts [Client\*235]:

• Consumer \*038: Also has ‘Customer’ as a synonym.

• Stakeholder \*233

• Client Stakeholder \*650

• Server Stakeholder \*651

Type [Client \*235]: Role [Stakeholder].

## Client Stakeholder \*650 May 21 2005

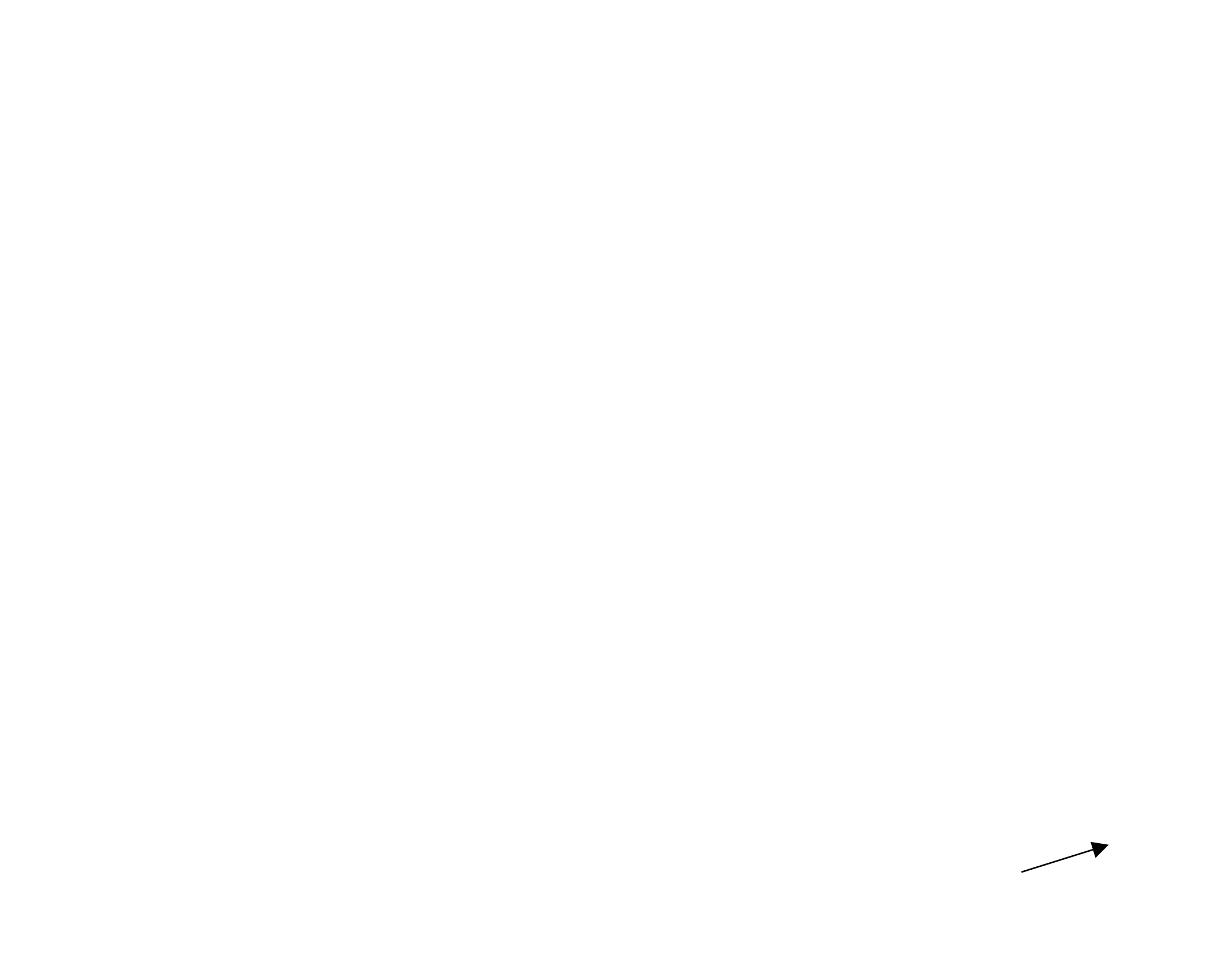
A Client Stakeholder states needs, approves requirements and receives benefits or results produced by a Server Stakeholder.

Related Concepts [Client Stakeholder \*650]:

• Server Stakeholder \*651

• Client \*235

• Stakeholder \*233

Figure \*650 The multiple levels of client stakeholders.

## Combine: Concept \*393. July 21 29th , 2001

Combine means: to put together, in some sense and manner.

Related Concepts: Combine, Compress, Condense, Summarize, Implode

Keyed icon [Combine, \*393]: ‘}’ (Icon). Antonym Icon: { Generate.

A } B means ‘A {Combine/Compress/Condense/Summarize/Implode} to B’.

*Examples:*

Project Data } Report A

IF Companies } One Corporation

Design Ideas } Design Decision

Conflicting Requirements } Balanced Set of Requirements

Target {Scale, Goal X, Fail X} } Gist. = Target } Gist

## Comment Concept \*018. January 8, 2003

see Note \*018

## Commentary Concept \*632 May 26, 2003

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

Examples:

• Note

• Comment

• "Text in quotes"

• Source

Related Concepts [Commentary \*632]:

• Non-Commentary \*294

• Core Specification \*633

• Background \*507

• Specification \*137

Type [Commentary \*632]: Specification.

## Commitment Parameters: Concept \*027.

July 21, 2001

Future requirements specification levels which are ‘promised’ at some level.

*These are distinguished from non-commitment target parameters:*

*•  Wish: valued, but not necessarily practical.*

*•  Ideal (‘perfection’) target specifications, which are ‘possible, desired, noted, but’ not promised to anyone’ targets.*

*•  Stretch, which is something we will try to look into, but is not promised.*

|  |  |
| --- | --- |
| *Commitment Target* Parameters | *Non-commitment Target* Parameters |
| *Fail* | *Wish* |
| *Goal* | *Ideal* |
| *Limit* | *Stretch* |

Rationale: this concept is needed so that engineers do not misunderstand all specifications as being a commitment. It also helps us understand the relative priority of a specification.

## Common Causes Concept \*020 November 21, 2002

These are repetitive systemic causes of consistent measurable process output levels of performance or cost.

“Common causes: of variation, produce points on a control chart that over a long period all fall inside the control limits. Common causes of variation stay the same day to day, lot to lot.” (DEMING93, p.178).

Dr. Shewhart (Deming’s teacher, who invented the concept, called them “assignable causes".

Note: The main point of this concept is that common causes are rooted in ‘organizational‘ things, which ‘must be improved by management’ (Deming’s contention), and are outside the capability of a worker to improve or deal with. The term common causes was first used in 1947 and published by Deming in 1956 (DEMING86 p.314).

Related Concepts:

• Special Causes \*019

• Statistical Process Control \*477

Synonym: assignable causes

Type: Statistical Concept

Domain: Statistical Process Control, Continuous Process Improvement

## Commonality [Product Line] Concept \*580 July 13, 2002

If any two products share any common assets they have commonality.

Note: in building a product line architecture you will try to maximize commonality potential by both the product line architecture itself, and the design of individual components and individual development artifacts.

See variability

*Thanks to Stefan Ferber, Bosch, for suggesting this concept. July 2002.*

## Competitive Engineering. Concept \*395.

July 21, 2001

A Systems Engineering discipline that is focused on ‘winning’, relative to some competitor, some enemy, some challenges, some events, or some Benchmarks.

Notes:

Competitive Engineering involves:

• analysis and specification of competitive Benchmarks.

• determination of competitive target levels for qualities and costs

• determination of competitive target qualifiers {when, where, if}.

• analysis and avoidance of undesired risk elements.

• rapid recognition of changes, and response to them, in the engineering specifications and project plans

• creativity in finding design ideas with maximum satisfaction of quality targets at minimum costs

• the ability to avoid, identify and solve engineering problems as early as possible

• the ability to invest in continuous data collection about product and engineering process, so as to sense problems, and deal with them on a factual rational basis.

Concept and definition © Tom Gilb 1999-2001

## Complete. Concept \*270.

July 21, 2001

A ‘complete’ specification, of anything, especially design or requirements, is not lacking in any information necessary for ‘successful’ use of it by [defined parties], at [defined times], under [specified conditions].

*Note 1. One formal and measurable (using the SQC process) notion of ‘complete’, is that no defects, according to all pertinent rules for the specification, exist.*

*Note 2. Another notion is that it is complete for a [defined purpose or situation], though possibly incomplete for other situations.*

## Complex (Adjective) Concept \*021 Aug 31 2014 Major redefinition Version 17:20, Sept 17 2014 previously March 31, 2003 B,

Compound in structure; consisting of interconnected parts.

Related Concepts [Complex \*021]:

Compound \*023

Elementary \*055

Antonym [Complex \*021]

Simple

Note

**Complex, according to Snowdon, Cynefin**

“A complex system is a system composed of interconnected parts that as a whole exhibit one or more properties (behavior among the possible properties) **not obvious** from the properties of the individual parts.[1]

A system’s complexity may be of one of two forms: disorganized complexity and organized complexity.[2] In essence, disorganized complexity is a matter of a very large number of parts, and organized complexity is a matter of the subject system (quite possibly with only a limited number of parts) exhibiting emergent properties.”

<http://cognitive-edge.com/library/more/glossary/#complexityNote> *[TG, August 31 2014.20:01]*

*I think this is a muddled definition. It mixes ends and means. It mixes the problem of lack of predictability, with some scattered and selected reasons why this maybe so. It is not logical. It is not a complete picture. It is not based on any research cited or shown here. It is not based on any practical applications cited or shown here. It does not give examples here, to help understand the terminology. It is unsystematic ramblings.*

*I have problems with many phrases: for example ‘not obvious’ (to whom, with which tools?).*

*Disorganized complexity =* very large number of parts *.* “organized complexity” = *“*exhibiting emergent properties”:

Where does he get this stuff from?

*I choose to NOT define Complex****ity*** *(the noun, attribute, degrees of with a Scale. If we do define it, it needs to be in terms of structure and interconnection, not intelligibility) in PL yet.*

*One point I am thinking about is whether you can have complexity from a single component, Elementary, system? Does it have to be compound? I don’t think so.*

*Is a ‘black box’ content compound? We don’t know, by definition, and don’t care. The ‘*very large number of parts*’, and ‘*limited number of parts*’ are not essential characteristics. A given black box is Complex or not. It exhibits ‘difficulty for us to understand and predict’, or not.*

*This can be due to a quantity of parts, or not, or to the nature of their couplings, or not. Or to the outside of the black box environment, or not. In short, unpredictable behavior may be due to one or more causes, not all of them necessarily known or perceivable to us mortals, at the present state of knowledge. Quantity of parts is one possible factor, but not a necessary factor. I think Snowdon may have inadvertently, & intuitively, given it unnecessary and undue prominence; thus leading to confusion. Even a single part entity, in a black box, may exhibit difficult-to-predict behavior, due to interaction with the outside world (think, heating or shaking, or sending waves to the box).*

*So, the essential attribute we are trying to describe is its unpredictability, and using the term ‘Complex’ is misleading, if the essence is that ‘it must be caused by multiple parts inside, interacting’ (which the term Complex implies). It is only valid if we distance ourselves, from the root notion (many parts) and focus on the ‘understood’ meaning of complex (difficult). In which case we should choose a purer term to use, and define that term independently of the quantity of parts, or of any other potential or partial case of the difficulty of understanding it. The lack of perceptibility in itself is of interest, and the causes are only secondary.*

*The term ‘complicated’ is better: my dictionary definition is “1.difficult to analyze or understand” (a note there in the dictionary “similar to : complex”*

## Complicated Concept \*676 Aug 31 2014

Anything, that is difficult to understand, estimate or predict the actions, reactions, and attributes of.

Notes:

1. it does not matter whether the Complicated thing is Compound or Elementary
2. **Complicated** (difficulty of understanding) can have many different and simultaneous causes; not just number of parts.
3. **Intelligibility** refers to specifications, and human ability to interpret them correctly, but not necessarily to ‘predict behavior for them or their consequences.
4. ‘Complicated’ is ***more general*,** ‘anything’, and it is about *all* systems, not just specifications. Complicated refers to our ability to understand the Complicated system (and its attributes) *as it is*, and possibly *as it will become*; to estimate current and future emergent properties (attributes it will display), to predict future behavior.

Related Concepts [Complicated \*676]:

Risk \*309

Uncertainty \*310

Compound \*023

Elementary \*055

Complex \*021

Intelligible \*675 see note 3 and 4 above, for contrast to Complicated.

Synonyms: [Complicated \*676]

Unintelligible

Unpredictable

Inestimable

Unperceivable

Imperceptible

Unobservable

Undetectable

Type [Complicated \*676]: Adjective.

## Component "component" Concept \*022 January 11, 2003

A component is a potential, or actual, part of some larger system, or specification.

Notes:

1. A component can be elementary or complex.

2. A component is any identifiable individual part of any set.

3. Product Line Components are final parts of a product or a product line generic asset collection (assets which can be turned into products).

Synonyms [Component \*022]:

• Element \*022

• Is Part of \*022

Related Concepts [Component \*022]:

• Elementary \*055

• Complex \*021

• Consists Of \*616

• Includes \*391 “Includes can give a partial list of contents.”

Keyed Icon [Component \*022]: X: {y}. “Meaning y is a component of X.”

Type: Specification Object.

## Compound "compound statement" : Concept \*023 August 31 2014

Compound entities contain two or more elements.

Notes:

1. One class of compound entity is multiple Planguage statements, grouped together, usually with a tag at the compound level.
2. Another class is compound system elements, of any and varied types, such as compound strategies or designs.
3. Requirements, Design Ideas and Evo Steps are often complex components.
4. If something can be decomposed into sets of other compound and/or elementary elements, then it is compound.

Related Concepts: [Compound \*023]

• Elementary, \*055 is ‘non compound’.

Example:

Goal [Alpha]: 30%, [Beta]: 20%. “This is a compound statement.”

Goal [Theta]: 50%. “This is an elementary statement.”

Edit notes: Major reedit (My PL ‘Complex’ was redefined (in the unintelligible direction), it was used in the compound sense) August 31 2014 . Initial creation November 20, 2001,

## Concept: Concept \*595. March 12, 2003

A concept is an idea.

Related Concepts: [Concept \*595]

• Planguage Concept, \*188 an Idea defined in Planguage.

• Engineering Concept, a notion of an engineering artifact such as a design idea or requirement idea.

Defined concepts can:

• be reused without explaining them again.

• be redefined by Planguage Users locally, and simultaneously changing (hopefully improving) the definition of all terms, which reference the defined concept.

• be 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.

Type [Concept \*595]:

## Concrete Strategy Concept \*661 August 13 2008 (Born)

A concrete strategy any device or means for supporting specified aims, which is existing in a real or material form, and is not abstract.

Notes:

This can include such real forms as: organizations, technology, resources.

But specifically excludes goals, aims, objectives, constraints and other abstract forms.

Real world functions with attributes which impact higher level objectives.

Related Concepts [Concrete Strategy \*660]: <- Vision Engineering paper Aug 13 2008

• Abstract Strategy \*660

• Strategic Objectives (\*214)

• Means Objectives (\*215)

• Strategy (\*047)

• Aim \*001

Type [Abstract Strategy \*660]: Specification.

## Condition Concept \*024 April 27, 2003

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.

‘[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 [Condition \*024]:

• Conditional Term \*024

• Pre-requisite Condition \*024

Related Concepts [Condition \*024]:

• Condition Constraint \*498

• Qualifier \*124

• Status \*174: The result of the evaluation of a condition.

Keyed Icon [Condition \*024]: [?]

Notes:‘[?]’ alone is a keyed icon meaning Condition and its synonyms. The ‘?’ is an icon for a condition or set of conditions whose status needs to be determined as true or not.

The qualifier square brackets alone ‘[ ]’ serve as a keyed icon for containing conditions, in the context of typical Planguage statements.

Type [Condition \*024]: Parameter, Qualifier Concept.

## Condition Constraint Concept \*498 June 4, 2003

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’).

Examples:

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 [Condition Constraint \*498]:

• Restriction \*498

Related Concepts [Condition Constraint \*498]:

• Condition \*024

• Constraint \*218

• Status \*174: Synonym is ‘State’

Keyed Icon [Condition Constraint \*498]: [<Condition Tag Name>]

“Square brackets. Same icon as for a Condition”

Type [Condition Constraint \*498]: Constraint.

## Confidentiality: Concept \*647 20 April 2004

The level of readership to which a specification is restricted.

Parameter [\*647] Confidentiality.

Type [\*647] Parameter

## Constraint Concept \*218 December 4, 2007

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.’

Examples:

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.

Examples:

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 [KEENEY92]:

• 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.

OK

Figure \*218: 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.

10. A requirement is a ‘constraint on succeeding engineering processes’ in fact, if and only if it has an authority, or other form of priority, which in fact 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.

11. On the dynamics of constraints (Dec 2007)

1. there is always a relevant and perhaps critical set of constraints, for any endeavor (project, organization, process)

2. the applicable set of constraints will change

3. the recognized set will change

4. our priorities that determine which constraints are applicable may change

5. the priority of a particular constraint may cause that constraint to be ignored (given lower priority than another requirement).

Related Concepts [Constraint \*218]:

• Requirement \*026

• Function Constraint \*469

• Performance Constraint \*438

• Resource Constraint \*478

• Design Constraint \*181

• Condition Constraint \*498

• Survival \*440

• Fail \*098

Figure \*218a: Drawn Icons for Constraints

Keyed Icons [Constraint \*218]:

For Survival: [ and/or ]

For Fail: !

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

Type [Constraint \*218]: Requirement Class, Parameter.

## Constraint Design: Concept \*523. January 24, 2002

Constraint Design is design specification aimed at, or as a side effect, satisfying Constraint requirements.

Note 1: On the multiple effects of any design.

Any design specification, however intended, in reality will impact many of our requirements. This multiple impact is true whether we like it or not. We cannot, it seems, only design on one pure requirement dimension (cost, performance, quality, condition constraint, function) without having some side effects in others.

Type: Design specification

Related Concepts [Constraint Design \*523].

• performance \*434

• design specification \*586

* Function Design \*521
* Performance design \*520
* Design \*046
* (performance) target \*048
* (performance) constraint \*218

• cost design \*522

• function requirement \*074

• performance {architecture, solution, design idea, and other synonyms of design)

## Consists of \*616 May 19 2005

Parameter:

Describes *complete* set of elements that a concept consists of.

Security:

Consists of : {Integrity, Attack}.

Related Concepts [Consists of, \*616]

• Includes \*391 may list some but not necessarily all components; Element \*022.

Keyed Icon: *=* { . . . } ‘‘Is equal to the set.’’

Example:

Core Family = {Mother, Father, Children}.

## Consumer Concept \*038 May 6, 2003

A consumer is a person or group, who makes use of (‘consumes’) a process output (product).

Notes:

1. A consumer is any customer ‘downstream’ in a chain of systems engineering or system implementation processes. Consumers mainly make use of intermediary process outputs of systems engineering, management and manufacturing processes, such as test plans and requirement specifications. A consumer at the end of the chain of consumers, who makes practical use of the final real system, is usually termed a user.

2. Use the term ‘Consumer’ for clarity on this term. The use of the synonym term, ‘Customer,’ is to remind employees that their colleagues in production or development processes, are to be treated like ‘customers.’

Synonyms [Consumer \*038]:

• Customer \*038

Related Concepts [Consumer \*038]:

• Client \*235: Also has ‘Customer’ as a synonym.

• User \*234

• Stakeholder \*233

Type [Consumer \*038]: Role [Stakeholder].

## Content: Concept \*594. August 15, 2002

Content is the meaning or message contained in a specification as distinct from its appearance, form, or style.

Actual specification content may or may not totally reflect a concept we have in mind. It might intentionally simplify the concept, or specify a particular view of it.

Related Concepts [Content \*594]

• Specification \*137, the formal documentation of a concept. Content is what is contained in a specification.

• Concept \*595 an idea, which may or may not be specified.

• Context: \*483 a limited view of the problem from any useful perspective

• Content Rules 543 the engineering standards for how to derive, specify and analyze engineering content, as opposed to clarity rules (\*129) which specify how to make an idea (good or bad) clear.

## Context. Concept \*483. July 20, 2002

A context is a system view from any useful perspective.

Example:

Life Cycle Context: {Being Produced, Being Tested, Being Delivered, Being Installed, Being Tested After Installation, Being Used, Being Supported, Having Waste Removed, Being Disposed Of}.

Source: Robert Halligan, Systems Engineering Course, 5 September 2001. [www.taa.com.au](http://www.taa.com.au). He says it is a common practice to do context analysis using context diagrams.

Usage:

Contexts are established to force us to consider requirements and design that we might otherwise forget.

Related Concepts:

• Stakeholder: \*233: stakeholders are a particular class of context. Contexts by contrast may have no real, active or responsible stakeholder. Consequently a context can be established to make sure that some critical view of the system in space or time is considered.

• View: \*484

• Context Diagram: a diagram of selected contexts or views of the system. (Halligan)

• Context Analysis: analysis of considerations, particularly requirements, in defined and selected contexts.

• qualifiers \*124: in Planguage, qualifiers is a major way of specifying a context.

Note: this use of context analysis is a risk management technique.

Type: systems analysis category.

ECB Emergency Communication Buoys

Sept 5 01 Workshop 3 Context Analysis.

Purpose of tool: to capture and validate requirements.<-RH

Water. Assume Sea?

SSE Submerged Signal Ejector. Assume x instances?

SA Sleeve Adaptor <-4.1.1c

Hull.

Air - Surface

Sunlight <-4.1.10

Oil <-4.1.10

Flotation Collar <-4.1.7

Buoyancy<-4.1.7

Integration <- 4.1.9, &.7

Energy <-4.1.11

Shelf Life <-4.1.12

Operational Duration 4.1.16 E

Markings <- (Battery, Buoy) 4.1.13, .16

Unattended Support or Storage <-4.1.14

Testing <-4.1.15, 4.3.3.4

Cooling <-4.1.5

Peripheral programming Equipment <-4.1.5

Damage <- 4.1.4

Warnings <- 4.1.16

==================== operational requirements

Loading In <- 4.2.1

Activation <- 4.2.1,

Discharge 4.2.2

Distress Transmission Activation <- 4.2.1

Homing Signals Activation <- 4.2.1

Operator programming <-4.2.2

Radiation <- 4.3.3.4

Example of initial brainstorming of potential context categories. The cross reference (<-) is to the requirements paragraph which mentions the context.

## Continuous Process Improvement Concept \*424 January 21, 2003

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

Abbreviation [Continuous Process Improvement \*424]: CPI.

Related Concepts [CPI \*424]:

• Defect Prevention Process \*042

• Statistical Process Control \*466

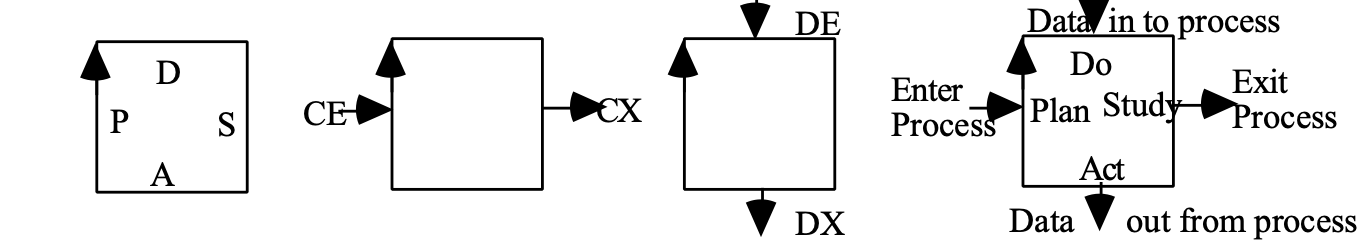
Type [Continuous Process Improvement \*424]: Process.

## Control Axis. Concept \*261. July 22, 2001

The Process Control Axis is the horizontal, left to right, direction of ‘process flow’ in a Planguage process diagram.

*Precedent processes are diagrammed as flowing into the left quadrant (CE below). The next process, after ‘this one’, is diagrammed as being pointed to by an arrow flowing out of the right quadrant (CX below).*

*Not coincidentally, the control axis coincides with the Plan and Study components of the quadrangle. See also Input/Output Axis.*



Process and data flow conventions in the Planguage Graphical Language.

These conventions can also be exploited in the Planguage Keyed Icons like this:

^[Architecture] 🡪 ^[Engineering] 🡪 ^[Component Design] 🡪 ^[Prototyping]

Showing the relationship between processes (^[Keyed Process Icon])

Below, we have added the notation for data in and out of the Architecture Process, the vertical axis.

Sys. Requirements

V

^[Architecture] 🡪 ^[Engineering] 🡪 ^[Component Design] 🡪 ^[Prototyping]

V

Arch. Specs

## Control Chart Concept \*029 May 18, 2003

“A control chart is a graphic comparison of process performance data to computed ‘control limits’ drawn as limit lines on the chart” [JURAN74

Source: The control chart was invented in 1924 by Dr. Water A. Shewhart, and was first expounded in his book, ‘The Economic Control of Manufactured Product’ Nostrand, 1931. It was first known as a ‘Shewhart control chart’, but extensive usage led to the shorter term ‘control chart’ [JURAN74].

Related Concepts [Control Chart \*029]:

• Control Limits \*028

• Statistical Process Control (SPC) \*466

Type [Control Chart \*029]: Feedback Data.

## Control Limits Concept \*028 May 18, 2003

Control limits are the upper and lower bounds on a control chart. Control limits should contain, within those boundaries, all observations of ‘variation due to common causes’. Observations *outside* the control limits are *probably* due to special causes.

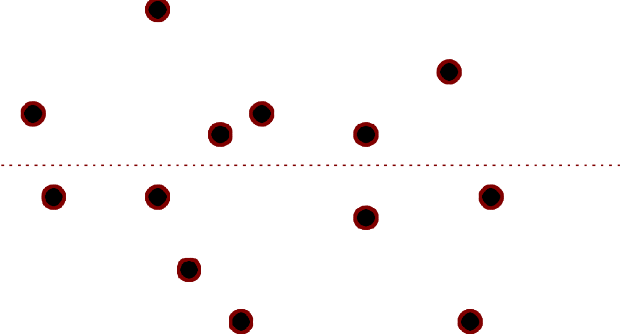
Notes:

1. Deming observed on control limits that:

“the device was invented by Dr. Shewhart in 1925 and used a ‘3-sigma’ control limit. They provide under a wide range of unknowable circumstances, future and past, a rational and economic guide to minimum loss from …. mistakes [in determining the distinction between Special and Common Causes]”.

[DEMING86, p. 319]

3-Sigma means plus/minus three times the standard deviation of the statistic used. It implies that only a tolerable 0.3% of observations outside the control limits are really due to common causes, and that they are also false alarms about ‘special causes’ [JURAN74, section 23-4].



A symbolic control chart with control limits being the solid horizontal lines. Common causes being within the lines and special causes outside the upper and lower bounds.

Related Concepts [Control Limits \*028]:

• Control Chart \*029

• Common Causes \*020

• Special Causes \*019

• Statistical Process Control (SPC) \*466

Type [Control Limits \*028]: Control Variable.

## Control Point Concept \*346 March 15, 2003 12:51

Control points are the specified elements that must be considered in solving a problem. They include requirements, constraints, qualifiers and inherited specification elements.

The number of problem control points that must be considered is a measure of the complexity of the problem to be solved.

Synonyms [Control Point \*346]:

• Problem Control Points \*346

• Problem Dimensions \*346

Related Concepts [Control Point \*346]:

• Problem \*270

• Design Problem \*654

• Problem Definition \*598

Type [Control Point \*346]: Engineering Concept, metric.

## Controllability "Controllability" : Concept \*175 January 21, 2002

A component of the system state vector is ‘Controllable’ if there exists a control signal which can bring the system from the current state to a desired state.

It is ‘Observable’ if it is possible to compute or estimate the state from a finitely long observation of the system output.

*Source: Steve Poppe, California, Personal Communication..*

*Concepts from feedback control theory, controllability and observability.*

## Controllable "controllable" . Concept \*232 July 22, 2001

‘Controllable’ is the ‘type of objective’, if the Design Ideas chosen are capable of influencing Fundamental Objectives "fundamental ojectives" at all.

*Source: Ralph Keeney. (see Bibliography)*

*Related Concept: Essential.*

*663*

## Core Ideology "controllable" . Concept \*664 Aug 16 2008 (Born)

Core Ideology consists of Core Purpose and Core Values.

*Source: Built To Last:* . Jim Collins and Jerry I. Porras. “BUILT TO LAST: Successful Habits of Visionary Companies”, Collins Business Essentials Pocketbook, 2002 edition, initial publication 1994

Related Concepts [Core Ideology \*664]

• Core Purpose \*662

• Core Values \*663

• Vision \*442

## Core Purpose "controllable" . Concept \*662 Aug 16 2008 (Born)

A Core Purpose is the most fundamental ‘why?’ the organization exists. A Core Purpose can be viewed as the most fundamental *objective*; the most fundamental *measure* of progress or improvement; the measure that all supporting strategies intend to improve. We must always aim to improve in that dimension, while keeping loyal to our core value constraints.

Part of the Core Ideology.

Core Purpose Examples [1, 4]

Sony: • To experience the sheer joy of innovation and the application of technology for the benefit and pleasure of the general public.

Merck: To preserve and improve human life.

All of our actions must be measured by our success in achieving this goal. [1, p.236. p.89]

Hewlett Packard: “To make technical contributions for the advancement and welfare of humanity”

Wal-Mart: “To give ordinary folk the chance to buy the same things as rich people”

Walt Disney: ”To make people happy.”

*Notes:*

Notice that the core purpose serves *two distinct areas*.

It defines a core **mission** (what we do) like ‘make people happy’.

But it also implies that ‘improvement is better’, within constraints (core values). So it gives a potentially measurable ‘**performance** dimension’. The more people made, happier the better. The more total happiness (people x degree of happiness) the better. The less pain or disease, the better.

*Source:* . Jim Collins and Jerry I. Porras. “BUILT TO LAST: Successful Habits of Visionary Companies”, Collins Business Essentials Pocketbook, 2002 edition, initial publication 1994*.*

*Related Concepts (Core Purpose \*662):*

*• Core Ideology \*664*

*• Core Values \*663.*

*• Objective and Performance Requirements \*100*

• Mission Concept \*097 (the ‘What’) Core Purpose is ‘Why’ for the ‘What’

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

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

Examples:

Core Specification Parameters include:

• Scale

• Meter

• Goal

• Definition

• Constraint

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 [Core Specification \*633]:

• Implementable Specification \*633

Related Concepts [Core Specification \*633]:

• Non-Commentary \*294

• Background \*507

• Commentary \*632

• Specification \*137

• Specification Quality Control (SQC) \*051

Type [Core Specification \*633]: Specification.

## Core Values "controllable" . Concept \*663 Aug 16 2008 (Born)

Core Values are a handful of basic ingrained beliefs about how to run a given organization. They are almost unchangeable for the lifetime of the organization. They are the strongest guidelines that all other activity in the organization must try to align with. They are the highest priority. The ‘core values’ can be viewed as **‘constraints’** on planning and activity. Anything that violates these constraints is not acceptable; is out of ‘alignment’ with the core values. Essentially they are constraints on reaching the Core Purpose.

Part of the Core Ideology .

*Source: Built To Last:* . Jim Collins and Jerry I. Porras. “BUILT TO LAST: Successful Habits of Visionary Companies”, Collins Business Essentials Pocketbook, 2002 edition, initial publication 1994

Core Values Examples [from ‘Built To Last’]

Sony Core Values: [Built To Last, p.237]

• Elevation of Japanese national culture and status.

• Being a pioneer – not following others, but doing the impossible.

• Respect and encouragement of individual ability and creativity.

Merck Core Values: [Built To Last, p.236]

Corporate social responsibility

Unequivocal excellence in all aspects of the company

Science-based innovation

Honesty and integrity

*Related Concepts (Core Values \*663):*

*• Core ideology \*664*

*• Core Purpose \*662*

*• Constraints \*218*

## Cost "cost" : (adjective). Concept \*185 January 21, 2002

‘Cost’ as an adjective is used to signify that ‘Resources’, rather than ‘Qualities’ or ‘Functions’ or ‘Designs’ are the category of the noun being used.

Example ‘Cost Constraint’: a requirement relating to cost, such as "budgets are maximum three million, unless Board approved".

More examples: Cost Attribute, Cost Requirement, Cost Level, Cost Plan, Cost Scale, and Cost Meter.

Note the essential distinction between the two similar terms:

• Resource: latent capability or fuels.

• Cost: expense incurred (resource used) in development or operation.

Examples:

So: here are some examples to help distinguish these two concepts.

Cost attribute: a system attribute relating to the expense of developing or maintaining.

Example: “ the calendar time currently necessary to develop a new language version of our base stations is three months”.

Resource Attribute: a system attribute relating to potential resources, such as total money, total time, total engineering hours, which are potentially available. These need to be distinguished from Resource Constraints (stakeholder-determined specified limits on the use of the potentially available resources; and Resource Budgets (Goal, Fail targets for specific qualifiers which regulate and plan our use of a resource, within the overall resource constraint.

Example: “this Corporation can deploy over 1 million engineering hours per year for new product development, if they are not misused by unnecessary rework”.

Cost Requirement: (Synonym: Budget)

Example: I hope we can now build the system on half the engineering hours we used for the last product in that line” (a Wish target),

Resource Requirement: a specified requirement that a defined resource be produced or be allocated, for potential later expenditure. This later exploitation would in the next process, be registered or tracked as a cost; a use of the resource.

Example: The system must produce 50 million barrels of oil annually”.

Cost Level: a numeric level on a defined Scale, measured by a defined Meter, probably expressed as a Benchmark {Past, Record, Stretch} of consumption of a resource.

Example: they used $2 Billion for a single ‘space launch’.

Resource Level: a numeric level on a defined scale of measure that is stipulated for production, allocation or is otherwise recognized.

Example: “all the gold on earth”.

Cost Plan: a specific Benchmark or Vision or Ambition specification regarding resource consumption.

Example: “ we aim to cut development time, to 10% of today’s time, within 3 years.”

Resource Plan: a specific Benchmark, Vision or Ambition specification regarding production or allocation of a defined resource.

Example: Our corporation will have to double current engineering resources and corresponding R&D capital available in order to remain competitive in the increasingly globalized marketplace of this decade.

Cost Scale: a defined scale of measure, units of measure, for costs or expenses.

Example: Scale: Engineering Hours per average Module delivered to Plan quality levels.

Resource Scale: a defined scale of measure, units of measure, for resources.

Example: Scale: Gigabytes actually free for deployment under [defined Circumstances] for defined [Purposes] after defined [Events].

Keyed Icon: -->O

Synonym: Resource (Adjective \*185, not to be confused with Resource noun \*199)

For example ‘resource constraint’, and ‘resource requirement’.

## Cost Concept \*033 May 26, 2003

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

Synonyms [Cost \*033]:

• Price \*033

• Expense \*033: The degree of consumption, how much resource was used.

Related Concepts [Cost \*033]:

• Resource \*199

Keyed Icon [Cost \*033]: -|->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:

--€--> to express a money cost

and ----<€€€€€€€€€|======> to express a cost range.

In a local culture, the local currency symbol can be used instead of the more neutral and default € symbol.

Example:

£, $, ¥, ---AUS$----> or ----CAN$--->

Type [Cost \*033]: Attribute Concept.

## Cost, to "Cost, To" : (Verb). Concept \*184 "\*184.Cost.Verb" . July 22, 2001

To ‘cost’ something is specifically to estimate the *resources* needed to achieve

a planned set of requirements. To estimate costs.

Type: process concept.

Synonym: To estimate (\*059). The term ‘to estimate’ is more general than ‘to cost’. ‘To estimate’ applies to *any* attribute or parameter estimate. For example, estimating the quality impact of a design.

Usage: They costed the system

In particular, the term ‘to cost’ implies that the estimates are about *financial* resources, unless otherwise specified.

## Cost Budget : Concept \*479. December 19, 2002

A cost budget is a requirement level set for a resource. Cost budgets are either Budget Constraints or Budget Targets.

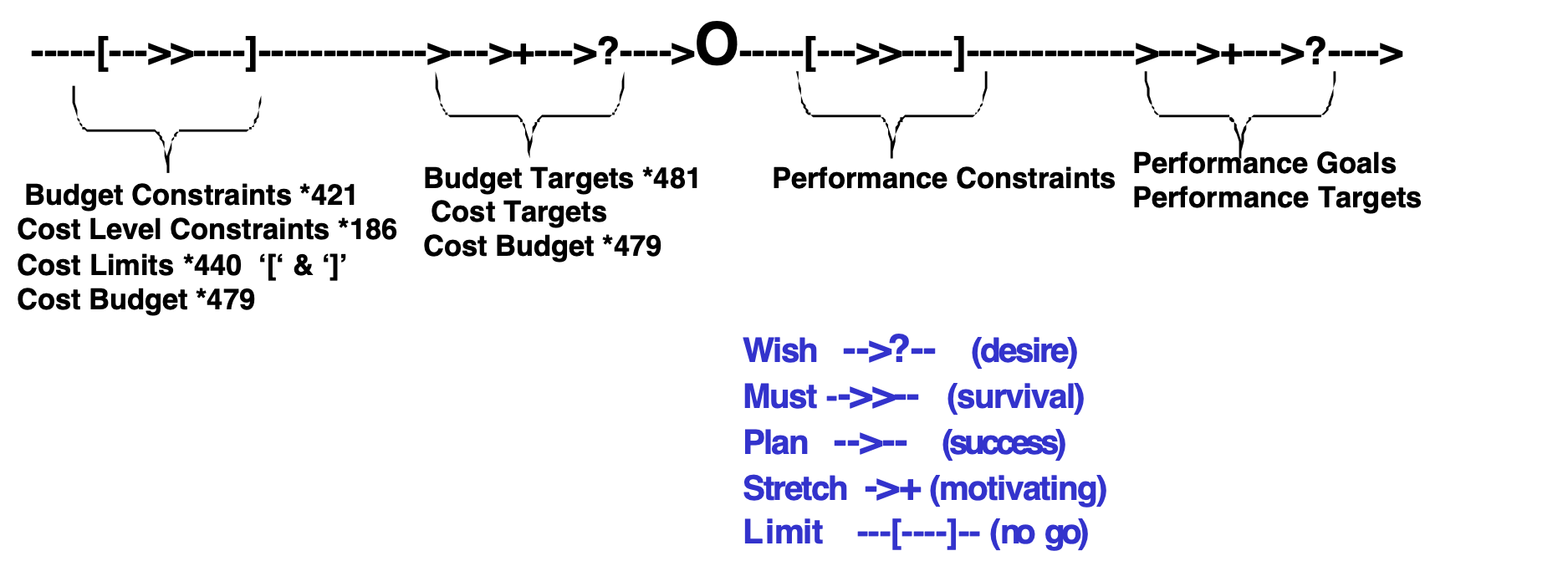
Related Concepts:

• Cost-Level Constraint: \*479 Limit and Fail for resources

• Budget Targets \*481. {Goal, Wish, Stretch}.

• Budget Constraints 421: these are Cost-Level Constraints and Binary Cost Constraints (\*467)

Type: requirement type.



EDIT NOTE DEC 19 2002 DIAGRAM NEED UPDATING TG

Synonyms:

• Cost Requirement \*479

Related Concepts:

• Budget Targets \*481

• Cost Target \*479

## Cost Design: Concept \*522. January 24 2002.

Cost design is conscious design specification aimed at, or at least incidentally resulting in, the achievement of specified cost targets and cost constraints.

Note 1: On the multiple effects of any design.

Any design specification in reality, however intended, will impact many of our requirements. This multiple impact is true whether we like it or not. We cannot, it seems, only design on one pure requirement dimension (cost, performance, quality, condition constraint, function) without having some side effects in others.

Type: Design Specification.

Related Concepts [Cost Design \*522].

* Performance Design \*520 Jan 24
* Function design \*521 Jan 24 2002
* Design (process) \*046
* cost target \*048
* cost constraint \*218
* cost {architecture, solution, design idea, and other synonyms of design)
* design specification *\*586*

## Cost-Level Constraint "Cost Constraint" : (Concept \*186) August 31, 2001 tg 1330

A ‘cost-level constraint’ specifies a cost-level limitation, on a defined resource scale.

It is a numeric cost requirement constraint, on the costs of design ideas, for a specified function or system.

A simple example of a cost-level constraint. Note that the Cost-Level Constraint is expressed by the ‘Limit’ specification. The ‘Goal’ specification concurrently expresses a ‘Budget’ (\*421), which is different.

Mission Budget:

Type: Cost Level Constraint.

Conditions [Non-NATO Missions Only, Peacekeeping Missions]

Scale: % of the deployed forces at any one moment of planned and actual deployment.

Goal [Next Budget Year, If Peace Conditions, Our National Contribution] 5%

Source [Goal]: Defense Budget for Next 5-year period.

Limit [Next Budget Year, If Peace Conditions, Our National Contribution] 10%

Authority [Limit]: NATO Ministers Meeting August 31 20xx, multinational agreed maximum restriction for any one nationality.

*Type: scalar resource constraint requirement*

*Synonym:*

*• Cost Constraint [\*186]. ‘Level’ implied, but not quite as clear to others.*

*Related Concepts [Cost-Level Constraint \*186]:*

• Cost Requirement. A cost-level *constraint* is a sub-class of ‘cost *requirement* ‘.

The ‘cost-level constraint’ is a fundamental requirement, handed to our project level by some defined level of stakeholder authority, which generally has higher priority than ‘our’ project level does. So, we must respect the cost constraint from the outset, and work within its framework. It is not something we can trade off , against performance requirements, and other resource requirements, without approval from the defined authority. Other cost requirement types (Budget Targets \*421) are set within the constraint framework and are more flexible as need, priority and resource availability permit. The cost constraint would always be set using Fail or Limit statements.

• Budget Targets (also ‘Budgets’), \*421:

The difference is that Budget Targets imply human intent to plan and budget for a limitation of resource use. Budget Targets imply willingness to ‘trade off’ these budgets with respect to higher priority requirements.

Cost-level constraints could normally be set due to other constraint causes than ‘human budgeting intent’; for example due to contracts, formal agreements, natural causes, legal limits, higher authority budgets or regulations and policies. Budgets by contrast to cost-level constraints (using Limit, Fail, Constraint) would always be set using Target statements Goal, Wish and Stretch.

• Resource Type Constraint, \*221

A resource constraint can also be a constraint on the *type* of resource used. Example:

• “Do not use capital, if current income can be deployed instead”.

• “Use local labor rather than offshore or imported labor.”

Resource Type Constraint is a Binary Constraint (Veto or Demand subtypes). Resource Type Constraints would not be stated in terms of a defined scale of resource. They would be binary in nature.

*Keyed Icon [Cost Constraint \*186]: -----[-->>----]-->O* (Resource *Limit icons)*

the ‘[‘ being a lower limit and the ‘]’ being an upper limit. The >> being a Fail type constraint specification. All of these can be multiple instances with different qualifiers for each one.

*Example (of multiple cost constraints in a single statement):*

Failure Levels: Fail [USA] US$ 100,000, [EU] ¤ 200,000, [Rest Of World, Next Year] GBP 300,000.

Illegal levels: Limit [USA] US$ 500,000, [EU] ¤ 500,000

*Cost constraints can be used as an overall framework within which a more-detailed set of Budget level specific constraints can be planned.*

*Example of global (applies entire project) and generic (must be determined in detail depending on profitability) cost constraint:*

C4:

Type: Cost Constraint:

Constraint [Project XX]: no expenditure shall exceed a level that will make the product unprofitable in the <short term>.

Authority: Business Policy 5.6.8

Parameters Used: Limit, Fail, Constraint.

## Cost Level "cost level" : Concept \*033. August 31, 2001tg 1510 May 6 03

Cost level is ‘expense’ level; resource consumption level.

A cost level specification is either a specific required (target), or historical (benchmark), for a cost attribute. It is a specific numeric value on the ‘Scale’, used to define the resource consumption level referred to.

Related Concept: Resource level. (The level a resource is currently at. The resource which remains after the cost expense is accounted for)

Synonym:

• expense level. (the degree of consumption, how much resource was used).

• Price \*033

Note: Current Resource level = Old level of resource minus Cost Level

"\*033.cost level.noun" Icon: -|->O (level ‘|’ of a cost/resource/system fuel ‘->O’)

Other specific attribute level indicators can be used as desired. For example >, <, >>, <<.

Usage: The ‘monetary cost of the project’ was more critical than the ‘cost of our reputation’, or the ‘cost in terms of future site rehabilitation - due to pollution’.

Example:

•  A statistical analysis system might have as a resource requirement, a certain quality level (accuracy, completeness, credibility) of data.

• This input data once had a set of other costs (time, people, money) needed to bring it to the quality levels required for some process. It is still the ‘data itself’ which is the fuel (resource) to the ‘statistical process’, not the previous-process costs of the data qualities. • The expense of the data quality (resource) levels, are the resources which were needed in a previous process, i.e. the process that made the data ‘good enough’ for the later statistical analysis process.

Note that a Cost level attribute, like quality and other performance attributes,

• can describe the ‘analyzed past’ of a system (benchmarks),

• and/or can specify future requirements (budget, targets).

The Cost level requirements

• can be specific target requirements

(‘The project budget [for Planned Levels] is ¤ 3 million’)

• or can be Global Cost Constraints

(‘No Budget can exceed ¤10 million without Board authorization 🡨 Policy’).

*Example:*

*Costs might be expressed like this:*

*‘about 1,000 engineering hours budgeted’.*

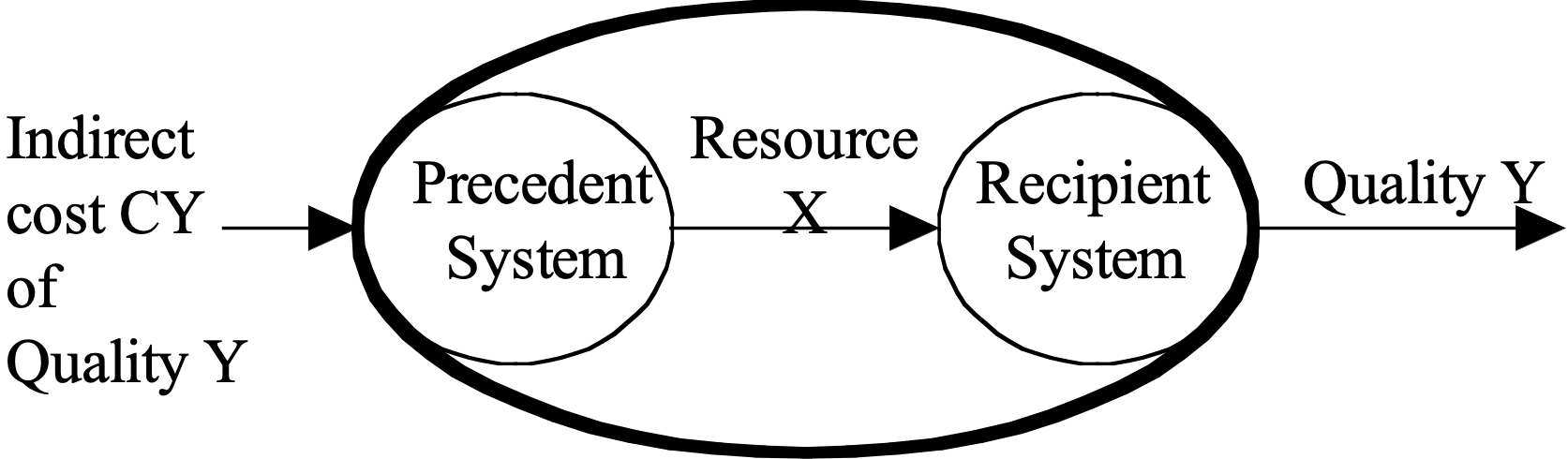
*Note that a ‘Cost’ is a numeric level of some defined cost type (its Scale defines the exact Cost type), like ‘investment $’. It is any type of Cost (or ‘Resource, or Process Input), work-hours, real time, talented people, goods, data and money.*

*A cost is not a Resource! It is a specific quantity (level) of a resource. A resource is a ‘potential source’ of cost inputs.*

*Note: a quality output (Resource X, below) from a precedent product or process, which itself is a necessary input to a subsequent system, is a cost (of operating the second system), and its costs (CY, in the first system) are the indirect cost of that quality.*



The indirect cost ‘CY’ of Quality Y is the cost of fueling the precedent system to the one producing the quality Y.



This is easier to see if we consider the two systems to be sub-systems of a single Supra-system, as above.

Related concept: System Input. The ‘cost’ (cost level requirement) is the level of System Input needed.

*Example:*

Capital Budget ------>>[1st Year]   3million --->O.

The double right-pointing arrowhead (>>), above, indicates a Fail (should not exceed) budget level of   3 million for the time period ‘1st Year’, on a scale (defined somewhere else in Capital Budget). This is a ‘cost budget’ type of requirement. The Attribute is identified by its tag ‘Capital Budget’.

Capital Budget --------------]  4million --->O.

The ultimate limit, a budget requirement type, is 4 million.

Capital Budget ------>>[1st Year]   3million -------------------]  4million ---->O.

This illustrates a combination.

Type [Cost Level]: resource requirements specification concept, budget requirement level

## Crash (parameter): Concept \*602. August 18, 2002

Crash is the popular and short form Planguage parameter suggestion for the concept Catastrophe Limit (\*602).

## Creativity Concept \*680 December 8 2014

The process of identifying, defining, and evaluating ideas, with respect to defined criteria.

The *degree* of creativity is the degree of effects on defined performance requirements, and on defined constraints.

1. Creativity is not mere spouting of ideas, and not mere ‘new’ ideas.
2. Creativity must have some defined purposes, or it would be meaningless
3. ‘Newness’ or ‘originality’ of the ideas is irrelevant, unless they are specific properties required.
4. When a set of ideas are specified, it should be possible to estimate the performance properties, and costs, that are expected, and compare these with requirements. This reflects the estimated degree of useful creativity.
5. When ideas are implemented, we can measure the real effects and costs of the ideas: the ‘real’ degree of creativity.
6. The creativity is sufficient, when all requirements are met, in practice.
7. Setting high or extreme requirements can be used to increase the creative motivation.

See paper: Gilb, T ; Practical Purposeful Creativity.

<http://www.gilb.com/tiki-download_file.php?fileId=22>

**Related Concepts** to Creativity \*680

Architecture \*192

Design Idea \*047

Design Engineering \*501

Design Process \*046 a synonym?

Innovation \*679

## Credibility Concept \*035 November 18, 2002, Note added 31May13

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).

Keyed Icon: ±?.# “An uncertainty estimate.”

Type: Systems Engineering concept.

Domain: Impact Estimation.

Related Concepts (Credibility \*035): Credibility Rating \*272, Evidence \*063, Source \*135

Note: *Some concepts of sources of credibility for evidence, as listed informally by Laurent Bossavit* [laurent@bossavit.com](mailto:laurent@bossavit.com) *May 31 2013 in a personal email (with kind permission), preparing for a talk at our London Conference.*

*“Grounding dimensions*

*- empirically grounded vs subjective opinion*

*- if empirical, availability of the original data (made up, scarce, paywalled, PDF)*

*- if empirical, sample size (amount of evidence: one-shot, replicated)*

*- money counts as sample size: billion-dollar claims often made in software engineering, economics (often overblown)*

*- source indirection: primary, secondary, higher levels*

*- semantic distortion from the original claim (increases with indirection)*

*- (distortion, indirect & repeated citation lead to "double counting" evidence)*

*- graphical interpretation from original data points*

*- mathematical rigor (or lack thereof: adding apples to horses)*

*- scientific integrity of source (fraud/plagiarists - sticklers)*

*- construct validity (absurd - problematic - well established - natural)”*

## Credibility Rating Concept \*272 February 11, 2003

A credibility rating is the assignment of a number on a defined credibility scale that ranges from 0.0 for no credibility for an impact estimate, to 1.0 for perfect credibility of an impact estimate.

Credibility Rating Meaning

0.0 wild guess, no credibility

0.1 we know it has been done somewhere

0.2 we have one measurement somewhere

0.3 there are several measurements in the estimated range

0.4 the several measurements are relevant to our case

0.5 the method of several relevant measurements is considered reliable 0.6 we have used the method/design/idea/strategy in-house

0.7 we have reliable measurements for the design idea in-house

0.8 reliable in-house measurements correlate to independent external measurements

0.9 we have used the idea on this project and measured it

(Evo step, pilot and field trial)

1.0 perfect credibility, we have rock solid, contract- guaranteed, long-term and credible experience with this idea on this project and, the results are unlikely to disappoint us.

Type [Credibility Rating \*272]: Defined Variable.

## Critical Factor Concept \*036 February 25, 2003/April 1, 2003

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

A critical *failure* factor is usually specified as a constraint level (for example a ‘Fail’ or ‘Survival’ level), or as a binary constraint (‘Constraint’).

A critical *success* factor is usually specified as a target level (a ‘Goal’ or ‘Budget’ level), or as a binary target. (‘Target’).

Notes:

1. System or component failure might occur, possibly only under certain ‘extreme’ conditions. These ‘critical’ conditions should be explicitly specified by some of the means available such as: {State, Condition, [qualifiers], Risk, Dependency, Impacts, Impacted By, Assumption, Authority, Source, and Impact Estimation specification terminology, such as Credibility, Safety Factor and others}.

2. A critical failure level of performance is usually defined by a Fail or Survival level specification with numeric level and relevant [qualifiers]. Each such level must be achieved or improved upon, otherwise critical breakdown, and *official* failure of the entire system, threatens to occur. Of course, reality might turn out to be different, either way!

3. Constraints (example: legal requirements or architectural requirements), for a product, are also generally ‘critical’. If they are not satisfied they might well cause failure somewhere, sometime, under given circumstances.

Related Concepts [Critical Factor \*036]:

• Critical Success Factor \*418

• Critical Failure Factor \*025

Type [Critical Factor \*036]: Classification Concept.

## Critical Failure Factor Concept \*025 April 1, 2003

A Critical Failure Factor is any factor that determines any degree of failure of a system. This includes, and can be specified and articulated using both scalar levels and binary conditions.

A critical failure factor may be specified as a constraint level (for example a ‘Fail’ or ‘Survival’ level), or as a binary constraint (‘Constraint’).

Related Concepts [Critical Failure Factor \*025]:

• Critical Success Factor \*418

• Critical Operational issues (COI) <- SPROELS02, DoD 5000-2-R [1996] Defense Acquisition, US Dept Def. Wash DC. “The issues (not parameters, objectives or thresholds) that must be examined… to evaluate/assess the systems capability to perform its mission.”

• Fail \*098

• Critical Factor \*036

Type[Critical Failure Factor \*025]: Classification Concept.

## Critical Success Factor Concept \*418 February 2 2007

Any an attribute level or condition of a system that can itself, *alone or in combination[[2]](#footnote-2) with other factors*, determine practical or total system success, is a critical success factor.

Related Concept [Critical Success factor \*418]:

• Critical Factor \*036

• Critical Failure Factor \*025

Acronym [Critical Success Factor \*418]: CSF

Type [Critical Success Factor \*418]: Classification Concept.

## Cumulative Defect Removal Effectiveness Concept \*037April 1, 2003 "\*037.cumulative effectiveness"

The cumulative defect-removal percentage, of a defined series of defect removal processes.

*For example, a series of specification defect-removal stages, plus various testing stages.*

*A practical example: Specification Quality Control has a single-pass effectiveness of 60%[logic bugs]-88%[interface specification defects] but cumulative logic bug removal effectiveness for a series of Specification Quality Control (SQC) processes has been noted at 95%, before any testing stages are necessary*

*Source: IBM and Sema Ltd. UK [in GILB93].*

*The term could also refer to any other forms of ‘cumulative effectiveness, outside of defect removal.*

## Cycle-Constraint: Concept \*039. July 23, 2001

A planning constraint on an evolutionary project cycle of any kind.

Note: Typically, expressed in a planning policy (example below), and requiring a cycle to be planned within certain time or financial limits.

For Example:

Two Percent Money Limit:

Type: Generic Budget Constraint.

Description: A result cycle shall use no more than 2% of total budget.

## Cycle Priority: Concept \*040. July 23, 2001

The calculated relative (to other potential cycles) importance of a specific planned Evolutionary delivery cycle.

Usage:

This is usually done using the benefit/cost ratio (also known as the quality-to-cost ratio) expected/estimated at the next cycle. This may be estimated using an Impact Estimation table. The higher the benefit/cost ratio, the greater the priority. Any specific interesting set of quality and cost factors, instead of all the specified factors, may be selected for a more-specific evaluation of priority.

*For example, the ratio of Usability to Capital Cost for [Next Years Deliveries, Europe] could be chosen to give priority to any step content best serving those narrower interests.*

## Data Concept \*319 April 17, 2003, minor edit (‘including’) 19 Aug 2010

Data is any form of signal, which humans or machines can *usefully* distinguish from other signals. Data is interpreted by some sensing agent, a reader, or a computer, which tries to convert it into useful information.

Data can be viewed as a necessary system resource. Data can also be viewed as a process input and as a process output. Data can be viewed in terms of its function (‘to warn’, ‘to give costs’), volume (bits), and in terms of both cost (cost to acquire, cost to store, cost to keep updated) and performance characteristics (including accuracy, updatedness, credibility, precision, correctness).

Notes:

1. Data is a primary form of input and output to intellectual, and computer-controlled, processes. Data includes {characters, symbols, words, expressions, statements, diagrams}.

2. Data is not random meaningless signals. It is organized for analysis, or for use to help make decision

## Damage Concept \*673 140814+200814

Damage is the negative consequences of successful Attacks to a system

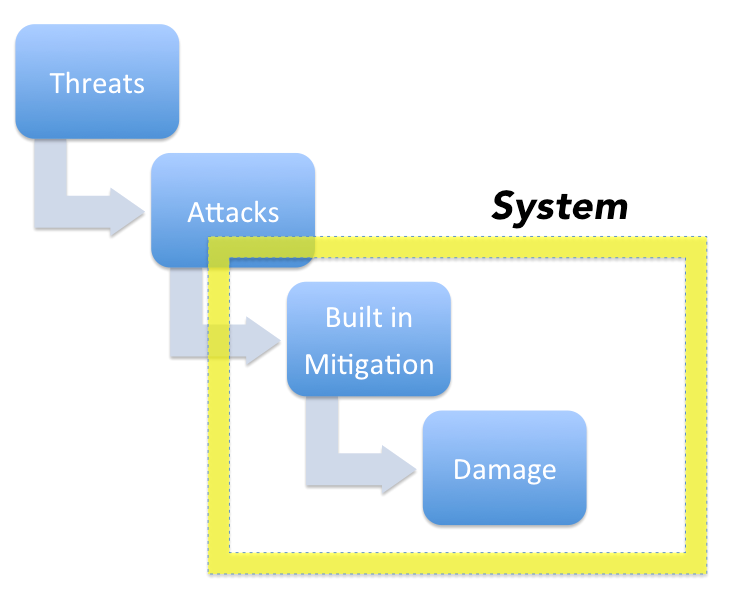


Figure \*673 A If built in mitigation fails to deal with attacks, some damage of some sort, by definition, results.

*Gross Damage: damage after built in Mitigation (attacks that succeeded)*

*Net Damage: remaining damage after Post Attack Mitigation (such as Insurance, Recovery)*

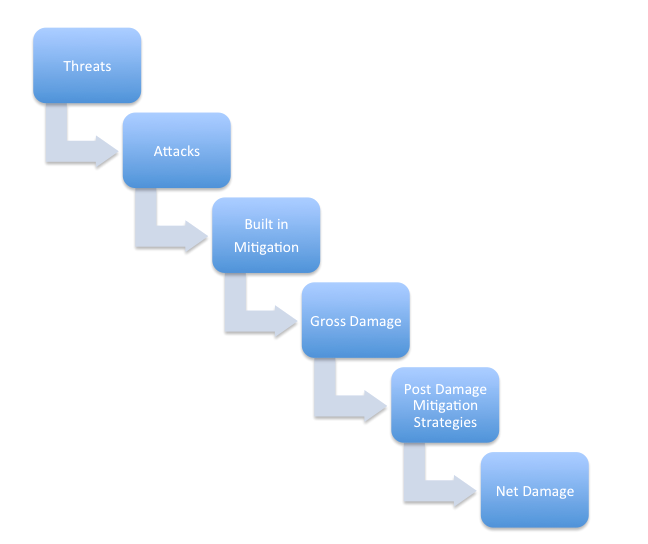


Figure \*673 B: the sequence of events leading to Net Damage.

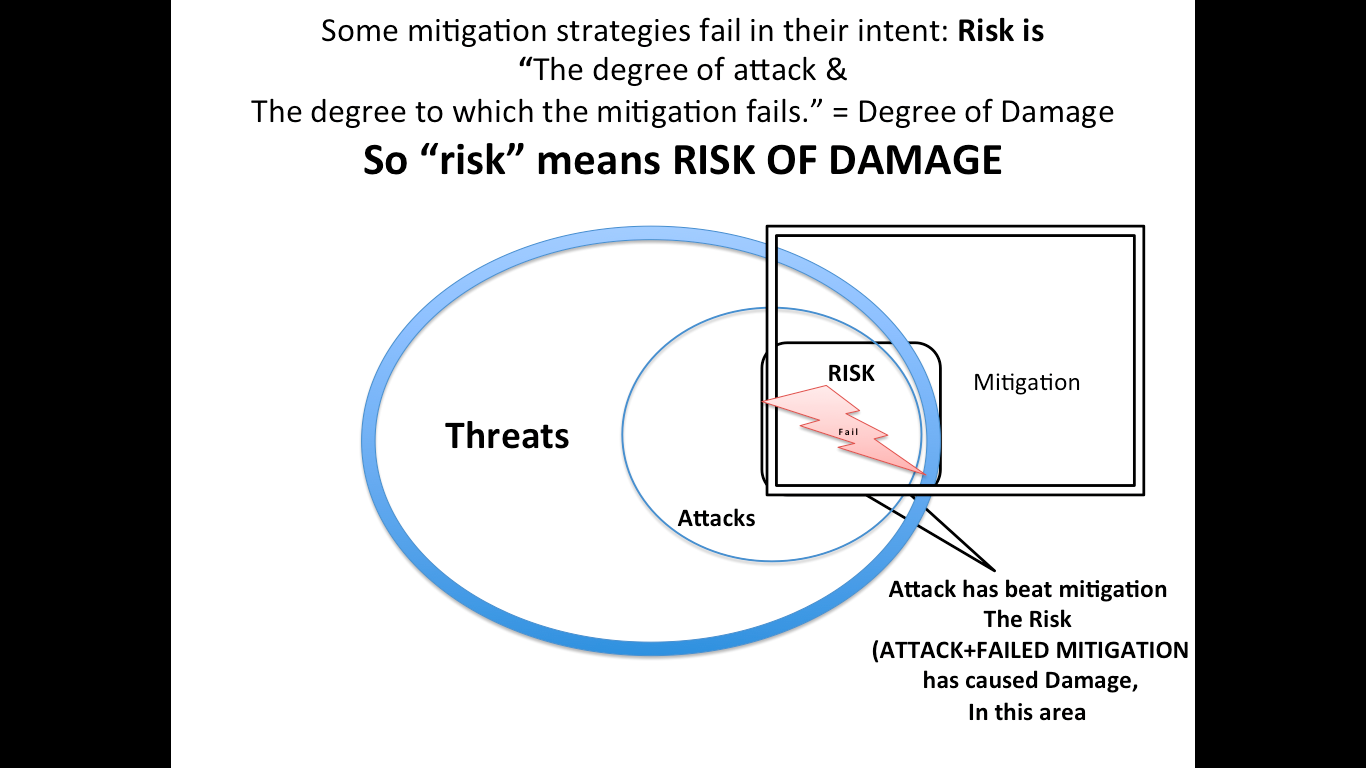


Figure \*673 C: The damage area viewed as the combination of attacks and failed mitigation. Risk means attack succeeds in causing damage in spite of attempted mitigation.

Notes:

*Related Concepts* [Damage \*673]:

*Integrity see CE book Chapter 5, maybe needs to be defined, maybe synonym for Robustness \*673*

*Threat: \*671.*

*Attack \*672*

*Mitigation \*670*

Keyed Icon [Damage \*673 ]:

Type [Damage \*673

## Dataware: Concept \*576 July 12, 2002

Dataware includes any data which can be or is in fact processed by a computer.

Related Concepts [Dataware, \*576]

• Planware, \*577 Planware is plans (information, data) for humans, not computers.

• Infoware, \*578 Infoware is information of any kind, a small subset of which might be used by computers

• Documentation, \*579 documentation is primarily for human not computer consumption

• Logicware \*575 is the ‘active’ logic directing computers in their actions. All Logicware is thus also dataware (since it is readable by a computer). But all dataware is not necessarily logicware.

## DDP Concept \*041 November 13, 2002

Abbreviation for Defect Detection Process.

## Debugging Concept \*293 June 5, 2003

A process of removing identified faults (bugs) from a system.

Notes:

1. A system is not mere specifications of itself, or a ‘mock up’ prototype. The ‘debugging’ process should include fault-consequent revalidation (in software called ‘regression testing’) that the fault ‘removal’ process has ‘succeeded’, and hopefully that it has not injected additional identifiable faults inadvertently.

2. The implication is that some QC processes on systems (for example, testing, field trials, user complaints, service/maintenance or repair analysis) have identified some faults, and ‘debugging’ means removing the faults, which were identified by that process. The debugging process is well downstream of the corresponding SQC processes of Editing and Edit Auditing, which deal with fixing defects (in specifications), not faults (in systems).

[Editor Note to Publisher: This Figure had to be assembled in parts as it could not be pasted special without crashing Powerpoint and running out of memory in Word?????? Figure in Powerpoint is complete – See Debugging \*293]

Type [Debugging \*293]: Process.

## Decision-maker Concept \*237 January 27, 2003

A decision-maker is a person or group, who will make a specific defined decision.

Synonyms [Decision-maker \*237]:

• Decision-taker \*237

Related Concepts [Decision-maker \*237]:

• Sponsor \*396

• Stakeholder \*233

Type [Decision-maker \*237]: Role [Stakeholder].

## Default Concept\*447 May 3, 2003

A default specification is a term to be used as a value when none is explicitly specified.

Notes:1. The rationale for a default is to save specification effort, and to simplify specification, where possible.

Example:

Scale: Time to <correctly> carry out a defined [Task, Default = Average Task].

which in application to this statement (which has no ‘Task’ specification qualifier):

Goal: 66 seconds.

would automatically mean the same as if you wrote:

Goal [Task = Average Task]: 66 seconds

But, you can override the default option with a statement like this:

Goal [Task = Difficult Task]: 100 seconds.

Synonyms [Default \*447]:

• Default Value \*447

• Default Option \*447

Type [Default \*447]: Specification Concept.

## Defect [Specification] Concept \*043

See Specification defect \*043

## Defect Detection Process (DDP) Concept \*041 November 20, 2002

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-focussed process of identifying and fixing specification defects.

Source: A detailed description of the DDP process can be found in [GILB93] and [WHEELER96].

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).

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”.

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.

Abbreviation: DDP.

Related Concepts:

• Specification Quality Control (SQC) (\*051)

• Defect Prevention Process (DPP) (\*042)

## Defect Prevention Process Concept \*042 January 12, 2003

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 on-going, 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 [DEMING86], 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 [MAYS95].

• A detailed description of DPP can be found in [GILB93: Chapters 7 and 17].

• DPP was the direct inspiration for IBM assessment process Level 5 (Ron Radice cited in Mays95), US DoD Software Engineering Institute’s Capability Maturity Model, CMM Level 5, and CMMI Level 5.

“I don’t expect perfection, but I do expect that we will always work for it. If we screw a mission up, that means that people who deserve to live will not live. That is going to happen. You know it. I know it. But we will avoid mistakes as much as possible, and we will learn the proper lessons from every one we make. Counter-terrorism is a Darwinian world. The dumb ones are already dead, and the people out there we have to worry about are those who’ve learned a lot of lessons. So have we, and we’re probably ahead of the game, tactically speaking, but we have to run hard to stay there. We will run hard.”

John Clark, Rainbow Six leader in Tom Clancy, Rainbow Six.

ISBN –0-14-027405-7, Penguin, www.penguin.com , page 37

[AWL EDIT NOTE: PERMISSIONS PENGUIN BOOKS LTD 27 WRIGHTS LANE, LONDON W8 5TZ, ENGLAND, MAYBE ADDRESS IS CHANGED TO 80 STRAND

ANYWAY RIGHTS ARE ADULTPERMISSIONS@PENGUIN.CO.UK••• REMOVE THIS NOTE, NOT FOR PUBLICATION]

Abbreviation [Defect Prevention Process \*042]: DPP.

Related Concepts [Defect Prevention Process \*042]:

• Plan Do Study Act cycle (PDSA) \*168

• Specification Quality Control (SQC) \*051

• Defect Detection Process (DDP) \*041

• Continuous Process Improvement \*424

• Process Improvement Suggestion \*088

• Process Improvement \*114

• Process Meeting \*119

• Process Change Management Team \*118

Type [Defect Prevention Process \*042]: Process.

## Define "define" (verb): Concept \*240. July 23, 2001 "\*240.define" .

To ‘define’ is to officially explain a specified concept in writing.

*Usage: In ‘Planguage’ the definition would always be represented (cross-referenced) by a unique tag. The tag would be capitalized (‘Tag’, not ‘tag’), both in the master definition, and sometimes when used elsewhere, to reference the concept or specification, as is practiced in this Glossary.*

*Rationale: Tag capitalization reminds us that it does have a formal definition elsewhere; in a Glossary or a project specification.*

*Usage: Defined terms in Planguage, as opposed to a project, would also have an asterisk ‘\*-number’ "asterisk \*number" associated with their concept definition, as does this concept \*240..*

## Defined Term: Concept \*151 (Term) 2/4/03

Same as Term

## Definition Concept \*044 May 25, 2003

‘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:

The following are equivalent definitions of the term, Trained:

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.

Examples:

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 [Definition \*044]:

• Def

Synonyms [Definition \*044]:

• Defined \*044

• Defined As \*044

Keyed Icons [Definition \*044]**: ‘:’**  *or* **‘=’**

“Whatever follows the icon symbol is a definition of the tag or parameter to the left of the symbol. ‘=’ is less commonly used.”

Type [Definition \*044]: Parameter.

## Degree of Target Fulfillment: Concept \*191. April 1, 2002

This is the degree, expressed as % of a [qualified] Target level (100%) , which it has been met, in relation to a defined Benchmark level (0%).

Synonym: (\*191.syn.target level fulfillment.) ‘degree of *target* level fulfillment’.

*Usage: The plan fulfillment can be real, estimated, or hypothetical. It is used in Impact Estimation .*

*For example:*

*• 50% means that half of the way to the target value is accomplished.*

*• ‘Minus 50%’ would be a corresponding distance (to +50%) in the opposite direction (a negative effect).*

## Delivered Value Concept \*371 January 14, 2003

Delivered value is the value delivered by a set of delivery steps to a stakeholder group, as a *consequence* of changing the system attributes.

Related Concepts [Delivered Value \*371]:

• Value \*269

• Delivery \*340

Type [Delivered Value \*371]: Metric.

## Delivered Level Concept \*533 January 3, 2003

For a scalar attribute, the Delivered Level is the numeric value on a scale of measure ‘delivered’ by an Evo step (that is, the resulting level after system enhancement).

A delivered level is a Past benchmark. It is a result of project measurement of progress towards a required Goal/Budget level.

Rationale: The reason for this term, instead of a simple ‘Past’ is to emphasize the series of Evo deliveries on a particular project, and distinguish them from other Past levels benchmarked in the project.

Example: O--<----•-------•[Step 22]------->------>

Example:

Availability:

Scale: % Uptime.

Delivered Level [Release 1.0]: 99.90%, [Release 1.5]: 99.95%.

Goal [Release 2.0]: 99.98%.

Related Concept [Delivered Level \*533]:

• Past \*106.

• Goal Step: ----)--- \*620

Keyed Icon [Delivered Level, \*533]:• “A bullet point”.

Type: Parameter, Benchmark.

## Delivery Cycle Concept \*049 January 2, 2003

A delivery cycle contains the initial operational implementation of an Evo Step, and its handover to stakeholders.

A delivery cycle is a subset of a Result Cycle. A delivery cycle involves implementation activities such as training, installation and field-testing.

Figure G.x: Diagram shows the component cycles of a Result Cycle.

Note: 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.

Delivery as a synonym of Delivery Cycle \*049

## Demand Constraint. Concept \*461. July 17, 2002 February 8, 2003

A demand is a constraint of type ‘You must do X’.

Type: non-scalar Constraint Requirement, parameter.

Domain: CE.Planguage.requirements.constraints.constraints.demand

Synonyms: ‘Do’ (Constraint), insistence, law, rule, regulation, non scalar constraint, a ‘must’, demand .

Related Concepts:

• Design Constraint \*181

• Veto Constraint \*462

Examples [Demand constraint]:

D1: Constraint: Resident workers in country of export shall be used wherever possible.

D2 Constraint: [IT Projects, In House] Commercial Off The Shelf Software shall be used exclusively.

D3: Constraint: Products and Services from our Corporation, Our customers and Partners are preferred. <- Corporate Policy 5.4

## Dependency Concept \*189 February 5, 2003February 5, 2003

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.

Examples:

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):

🡪 = ‘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 [Dependency \*189]: To promote awareness of relationships, ensure more realistic planning, and provide the ability to identify reliance, and therefore cope with any associated risks.

Synonym [Dependency \*189]:

• Depends On \*189

Related Concepts [Dependency \*189]:

• Before \*312

• After \*313

• Impacts \*334

• Impacted By \*412

Type [Dependency \*189]: Parameter [Relationship].

## Depends On: Concept \*189. July 24, 2001

This parameter expresses an *explicitly* defined dependency.

Type: Planguage parameter.

Synonym: Dependency (which can be used as the Parameter if you prefer).

Format:

A: Depends On: B

If B is not true then A is not true.

A Tag.

Dependency XX. “A Tag Depends On XX”

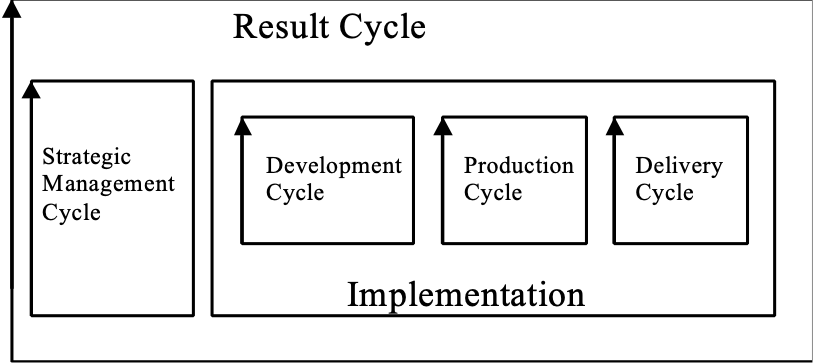
*Example:*

*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’.

*‘Implicitly defined dependencies’ use qualifiers, rather than explicit parameters like Depends On.*

## Deployment Concept \*358 March 8, 2003

‘Deployment’ means *actually delivering* an Evo step to a set of stakeholders. Deployment is the last stage of a Delivery Cycle.

*Figure [Deployment \*358]: Deployment is the last stage of the Delivery Cycle.*

Notes:

1. An Evo delivery cycle can consist of these tasks:

• Planning the delivery cycle including deployment

• Training Evo step recipients ready to use the upgraded system

• Upgrading the on-site system

(for example, upgrading software, database and/or hardware)

• Upgrading of supplier support activities to support the upgraded system

(such as help desk, maintenance services, spare parts supply)

• Carrying out field trials of the upgraded system, before scaling up and/or ‘going live’

• Evaluating field trials, and making decisions about full-scale deployment

• *Deployment*: Handing over to users/customers/stakeholders, operations, data gathering: achieving full scale deployment

Note, these are just examples. The actual delivery cycle tasks depend on exactly what is being deployed, the scale of the deployment, and the risks involved.

Related Concepts [Deployment \*358]:

• Delivery Cycle \*049

Type [Deployment \*358]: Process.

## Derived (adjective). Concept \*449 June 20, 2004

Something resulting from processing other information according to defined rules.

Note 1. A derived requirement is a specific requirement obtained by a generic requirement applied to a local range. For example: if a Generic Constraint says: we shall always be better in reliability than competitors, then the exact specification, for example:

Reliability:

Scale: Mean Time to Fail.

Past [Competitor X, Current Product] 20,000 hours.

Goal [Our New product] 30,000 hours.

The Goal level is the derived requirement.

Notes:

1. “Each solution brings **derived requirements** coming from the choice of the solution itself. For instance, vibration and heat are derived from a solution using a diesel engine, since an electrical engine would not vibrate but would rather create an electro-magnetic field.”<-Jean-Philippe Larat, [info@sodius.com](mailto:info@sodius.com) June 2004 INCOSE Tutorial.

2. He uses the apparent synonym ‘derived impacts’ which are a bi-product of the sequence ‘design analysis’ then ‘solution investigation’ then ‘derived impacts’ ( repeated as necessary). .”<-Jean-Philippe Larat, [info@sodius.com](mailto:info@sodius.com) June 2004 INCOSE Tutorial.

## Derivative Source. Concept \*450 July 30, 2001

The source of the Derived (\*449) specification or plan.

Synonym:

• Derivation, 450.

Warning this term (derivation) should be avoided because it can also mean the Derivative (452) and the act of deriving. It is uselessly ambiguous!

Note1. The ‘source’ implies the resource and the rule for conversion into the derivative.

## Derived Constraint. Concept \*451. July 30, 2001

A specific local constraint derived from a Generic ‘template’ Constraint or Global ‘wide scope’ Constraint.

## Derivative. Concept \*452. July 30, 2001

The thing derived from its Derivative Source. (\*450).

Note 1. Source is the initial information or resource, and some rules or process for derivation.

## Descendant. Concept \*268.Descendant July 25, 2001

An artifact descended from another artifact; immediately, or via several levels of process.

*A Kid is always a descendant of a Parent. A design could be the descendant of a set of requirements*.

## Description Concept \*416 18 April 2004

A description is a set of words and/or diagrams, that describe, and *partially* define, a component.

The parameter ‘Description’ (or synonyms, Detail, Detailed 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’.

Synonyms [Description \*416]:

Detail [\*416]

Detailed Description [\*416].

Type [Description \*416]: Parameter.

## Design Constraint Concept \*181 May 5, 2003

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).

Examples:

================ 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 [Design Constraint \*181]:

• Architectural Constraint \*181

• Design Restriction \*181

• Strategy Constraint \*181 (added 220814, CP book 8.1)

• Constrained Design: Informal Use

• Required Design: Informal Use

• Solution Constraint: Informal Use

Related Concepts [Design Constraint \*181]:

• Constraint \*218

• Requirement \*026

• Design Idea \*047

• Condition Constraint \*498

Keyed Icon [Design Constraint \*181]:

[<Design Idea Tag>] “The same icon as for a design. Alludes to a rectangle. [\_\_\_\_].”

[Editor Note to Publisher, the underline MUST be included in the above term with Design Idea tag. “Design Idea Tag” IS ALSO A USER DEFINED TERM, LIKE OTHER UNDERLINED TERMS]

Drawn Icon [Design Constraint \*181]: A rectangle within square brackets.

Type [Design Constraint \*181]: Constraint.

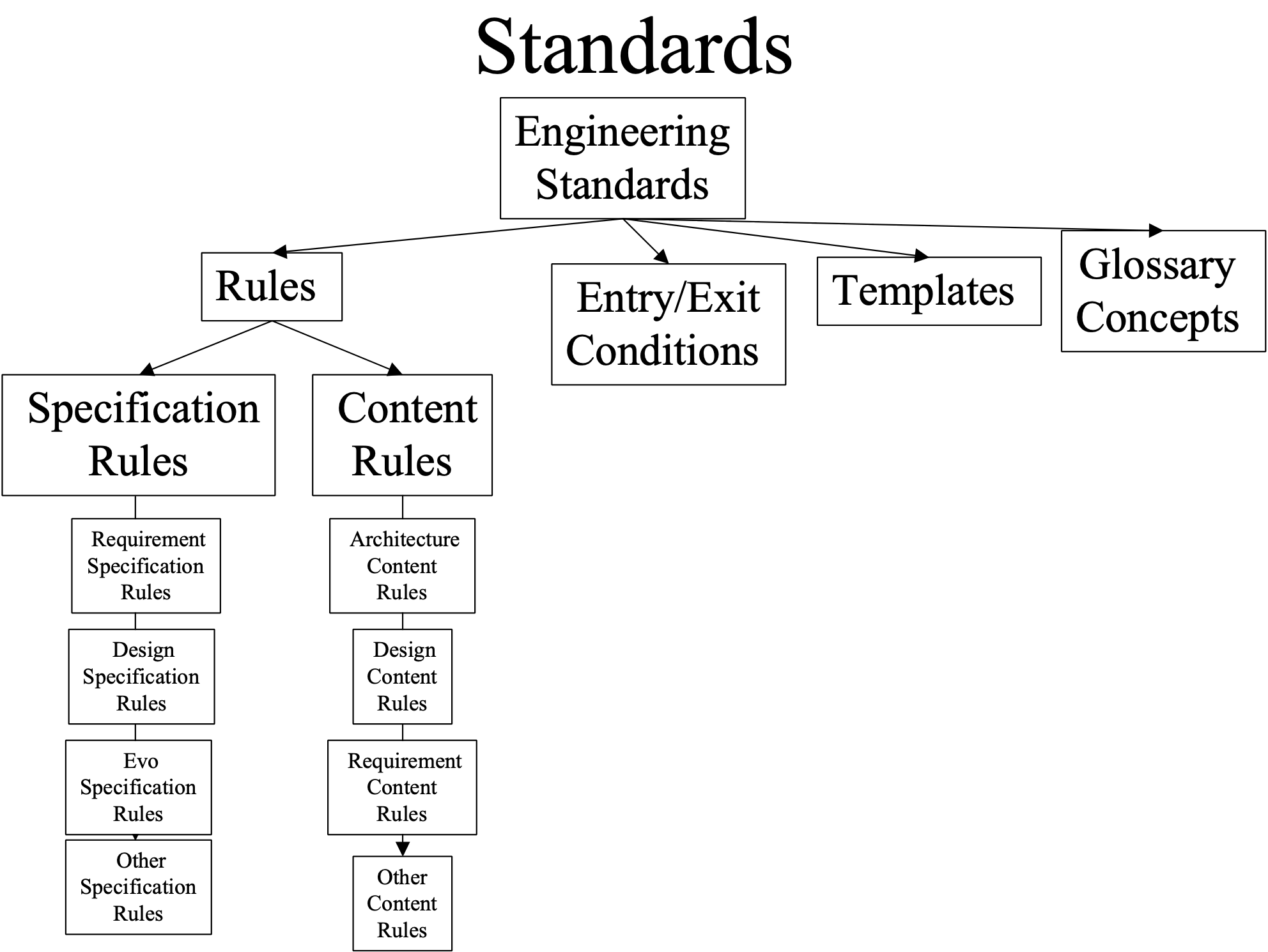
## Design Content Rules \*593 July 20, 2002

Design rules state expected design practices.

Design (content) rules give advice on best practices, and recommended practices, when designing (or architecting or design engineering, or finding means or strategies). Deviation from design rules should be the basis for a design review.

Unjustified violation of design rules are classified as design *defects*. These should not be confused with design specification defects, which are about how clearly a design is written up – not about its content or essence.

The density of design defects should be the basis for quantification of design quality. It should also be the basis for Entry to other processes of a design spec, and Exit from design processes of a design specification.



Related Concepts [Design Rules, \*593]

• clarity rules [Design Idea] \*129 rules about how articulate a design idea

• design specification \*586 – the spec which needs to be both written and judged according to the design rules.

• review criteria \*541 – Design Rules are one class of Review criteria

## Design Engineering Concept \*501 July 18, 2003

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).

2. The design *engineering* process can be subdivided into the following sub-processes:

a. *design requirements entry*: all relevant requirements to the design task are complete, clear, quantified, testable, validated by stakeholders, quality controlled, reviewed, and have thus obtained formal entry into the design process.

* 1. *search and acquire candidate design ideas* and their expected performance and cost attributes, along with any constraint threatening information. The search is based on matching design attributes with requirements, roughly
  2. *pick a reasonable set of design ideas*: this is done using Impact Estimation type of logic. It is done with regard to design attribute credibility, with regard to design attribute uncertainty, and with regard to necessary safety factor. When necessary, promising but uncertain, designs might need to be analyzed in depth using experiments, research, evolutionary implementation, and other devices to get more certain data about their attributes.
  3. *Deliver designs evolutionarily*: the candidate set of design ideas can then be delivered in an evolutionary sequence, one or more at a step, in priority sequence (high value to cost first) measuring actual results of performance and costs, until targets are reached, or budgets exceeded. Superfluous design ideas, in terms of reaching targets within constraints, can be kept in reserve for meeting improved target levels. But they do not necessarily have to be implemented when agreed targets are in fact reached.
  4. *Post delivery design improvement*: after a design is successfully implemented and delivered to some stakeholders, there is still the possibility that it might be replaced later by a later emerging superior (value to cost) design. This can happen before major project completion, or after initial system or project launch.

Related Concepts [Design Engineering \*501]:

• Design Process \*046

• Architecture Engineering \*499

• Systems Engineering \*223

• Engineering \*224

Type [Design Engineering \*501]: Discipline [Systems Engineering].

## Design Idea Concept \*047 June 13 2012 (to some degree, note 5), 2019 +\*215

A design idea is a

concept that is intended to satisfy some requirements, to some degree.

A *set* of design ideas is usually needed to solve a ‘design problem’.

Notes:

1. A *design specification* is a *written definition* of a specific design idea. A template for design specification is given in Figure 7.6.

2. A design idea is not usually a requirement. 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.

Requirements are *inputs* into a design process; design ideas are the *outputs*.

Note: A design can be a requirement if it is a *design constraint*. That is, a specific design is stated as mandatory or prohibited in the requirements.

3. 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 Goals levels, the design idea can be used.

4. <A design may be described in terms of its attributes. Design attributes can be function (and sub-functions). Performance attributes, costs, technical specification.

5. ‘One mans design becomes another’s requirement.’. The concept of requirement and design is *relative* to the level of process observation. A top level design becomes a ‘requirement’ at the next level

Synonyms [Design Idea \*047]:

• Design \*047, 'a design’

• Strategy \*047

• Proposed Solution \*047

• Means \*047

* Tactic \*047

Related Concepts [Design Idea \*047]:

• Architecture \*192

• Policy \*111

• Design Constraint \*181

• Design Specification \*586

• Design Target \*338

• Design Problem \*654

• Entity \*645

* Function Design \*521
* Means Objective (R Keeneny) \*215

Keyed Icon [Design Idea \*047]: |\_\_\_\_\_\_\_\_|

S1 S2

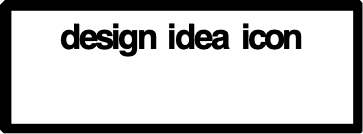
|\_\_\_\_\_||\_\_\_\_\_\_\_|

O-------[---🡨------🡪---]----🡪-----🡪 P

M Vb B T1 Vb T2

Illustration for paper ‘Vision Engineering’ (July 2008). M = Mission, Vb = Core Value barrier – constraint. B = Benchmark level. T1 and T2 are Target levels of Performance attribute P (a Core Purpose). S1 and S2 are Strategies. Based on ideas in Collins and Porras, “Built To Last”.

Graphic Icon [Design Idea \*047]: A lying-down rectangle. (The standing rectangle is a document icon.)



Type [Design Idea \*047]: Parameter, Design.

## Design Problem "Design Problem" : (Concept \*654) November 13 2005

A design problem is expressed as a requirement specification that includes a ‘complete set’ of valid targets and constraints, together with updated information about the degree of satisfaction in the design or the implementation. A design problem is the ‘set of gaps’ (to meet required levels) that remain to be designed for.

Note 1. We assume that the designer does not yet contain or have access to sufficient design solutions to reach all their targets, within all specified constraints, otherwise there would be no ‘problem’ remaining.

Note 2. Under Evolutionary Project Management, the design problem will evolve as the project progresses. Each new design idea, when implemented at an Evo step, will have a set of cumulative impacts on different requirements. The then remaining set of unsatisfied requirements (‘gaps’) become the new ‘design problem’.

Note 3.

On the notion of a ‘complete set’ of requirements in problem definition.

A simplified design problem could be formulated using one or a few requirements or constraints, just to see if that were technically or economically feasible at all. But the real design problem must be solved using all valid requirements. If this is impossible, then perhaps some of the requirements need to be realistically ‘relaxed’ to allow some of the others to be satisfied at all.

Short form: Problem.

Keyed icon [Design Problem, \*048]: -----<===>---> (The gap between < a past and > a target is an iconic symbol of a design problem)

O------------<-----------|=====The Gap===>------->

The progress from the Benchmark (<) to the current level (|) leaves a Gap (==) to the Target (>). This illustration brings in the idea that there has been some progress (‘<-------|’) since initial requirements were specified (‘-<-‘) , in delivering some degree of requirements target satisfaction using some design ideas. The gap (‘|====>” represents the remaining problem with this particular dimension of a requirement.

Synonyms: Design Target, residual system requirements, design gap, gap \*359 (a single instance of a gap amongst many which constitute the total design problem).

Related Concepts [Design Problem \*654 ]:

• Control Point \*346

• Problem \*270

Type [Design Problem \*654 ]:Engineering Concept.

## Design Process Concept \*046 July 18, 2003Btg

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 & Specify Design Ideas

• Evaluate the Design Ideas

• Select Design Ideas & 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.”

Blackmore in ‘The Meme Machine’ (See page 205)

Notes:

1. The design process can be subdivided into the following sub-processes:

a. Acquire stakeholder requirements.

b. Identify candidate designs

c. Tailor candidate designs to current purpose

d. Decide on a finite set of designs.

e. Do design review.

f. Implement designs in development system and test their validity.

g. Revise Design if necessary.

h. Implement designs in final system.

“Heuristic: At some point in the project, freeze the design. …..

This rule of thumb recognizes that a point is often reached in design where the character of a project, and hence the appropriate allocation of resources, changes from seeking alternative solutions to perfecting a chosen solution. …. After this point is reached, a major design change runs the unacceptable risk of introducing a fatal flaw because insufficient resources remain to evaluate all of its ramifications.” Source: Koen03, p.35

This quotation refers to b. (Identify) and c. (Tailor) above. It reminds us that the design process is not merely one of finding a design, and analyzing it. The selected design is likely to be generic in nature, or was applied in a different set of circumstances than the current project. So, we must expect to ‘perfect the chosen solution’ so that it better fits the current requirements and environment.

2. The reader might like to compare this simple design process with the corresponding design *engineering* process (\*501 note 2). The design engineering process is characterized by rigorous use of quantified information. It takes a clearer position on ‘how’ to design. The design process decomposition above specifies ‘what’ is to be done, but does not specify ‘how’.

Related Concepts [Design Process \*046]:

• Design Engineering \*501

• Systems Engineering \*223

• Engineering \*224

Type [Design Process \*046]: Process.

## Design Specification Concept \*586 May 29, 2003B

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.

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). These design specifications might also contain information about their expected attributes 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 \*586]: Design Specification Template.

Abbreviations [Design Specification \*586]:

• Design Spec \*586

Synonyms [Design Specification \*586]:

• Technical Design: Informal use

• ‘The Design’: Informal use

Related Concepts [Design Specification, \*586]:

• Design Engineering Process \*501

• Design Idea \*047

• Systems Architecture \*564

• Architecture \*192

• Architecture Specification \*617

• Specification \*137

Keyed Icon [Design Specification \*586]: []->@

This icon can be interpreted as a specification, ‘[]’ which impacts, ‘->’ target, ‘@’.

Type [Design Specification \*586]: Parameter, Specification.

## Design Target "Design Target" : Concept \*338 March 15, 2003

A Design Target is the specific target *level,* including *qualifier,* of an attribute requirement that is being worked towards, in the design or implementation process.

Usage 1. The designer uses a set of such design goals, including budget targets, as well as constraint information, to decide if they have found good enough design ideas to satisfy the performance requirements, within the constraints specified.

Keyed icon [Design Target, \*338 ] : ----> <level spec> [<qualifiers>].

The target level plan (or other target symbols) symbol, coupled with the qualifier

Example:

O-------------------->6,000 Hours MTBF [Release 6.0, US Market]----->. Reliability

Example:

Fail [End Next Year] 30. “a design constraint”

Goal [Within 6 Months After Launch] 60. “a design target”

The target is not merely the ‘60’ but includes the [qualifier] information.

In addition, the Fail and Goal specification both give critical information about the priority of the target (design survival point (Fail) and success point (Goal)).

*Usage 2. It is normal engineering practice for a designer to apply a performance target level with a built in safety factor, as the level they try to design for; for example two times the nominal specified level might be applied as the safety factor target. The Impact Estimation process makes use of this safety factor concept.*

*Usage 3. A design target may be specified using one of the target specification parameters {Goal, Stretch, Wish}. This applies to performance and budget targets.*

*Example:*

*Stretch [Next Release] 66.5 hours.*

*Related Concepts [Design Target \*338]:*

*• Benchmark \*007*

*• Performance Target \*439*

*• Qualifier \*124*

*• Budget Target \*481*

*• Condition Constraints \*498*

• Design problem, \*048

• Scalar target \*470

Type [Design Target \*338]: Requirement Level, Target

## Design To Cost Concept \*472 July 17, 2003

‘Design To Cost’ is an engineering strategy of consciously selecting design ideas, which will allow you to stay within any and all resource budgets.

Related Concepts [Design To Cost \*472]:

• Design Process \*046

Type [Design To Cost \*472]: Design Strategy, Process.

## Designer Concept \*190 May 6, 2003

A designer is a person or group who specifies design ideas, in an effort to satisfy specified requirements.

Notes:

1. This is a generic term. More specific terms include systems engineer, planner, or systems architect.

Synonyms [Designer \*190]:

• Systems Designer \*190

Related Concepts [Designer \*190]:

• Systems Architect \*193

Type [Designer \*190]: Role.

## Development Cycle Concept \*413 December 27, 2002

A development cycle is an optional backroom cycle, within the result cycle of Evo. It is 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.

Type: Process.

## Deviation Concept \*475 November 7, 2002

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.

Synonym: Variance.

Keyed Icon: ±

Type: Systems Engineering concept.

Domain: Systems Engineering.

## Diagram "diagram" : Concept \*228 "\*228.diagram" . July 26, 2001

A set of icons "icons" , symbols "symbols" and terms "terms" , which pictorially explains how something works, or tries to clarify the relationship between parts of the whole.

## Dialect: Concept \*204. July 26, 2001

A ‘Dialect’ is a local variation in Planguage symbols or terminology due to local culture, habit, traditions or language variation.

*Local culture can be a project, a division, a company, a charity, a national language group: any group which feels the need for variation or specialization in relation to a defined set of Planguage.*

*Type: Planguage class.*

*Synonym: Planguage dialect, local (Planguage) dialect, project (Planguage) dialect*

## Direct Constraint Concept \*510. November 21, 2001

Any constraint which is specified in the *format of a constraint*, and is intended to directly constrain performance, function, costs or design.

Type: Constraint.

Domain: Requirement.constraint.direct.

Related Concepts:

• Indirect Constraint, \*511. Any other (than direct constraint format) specifications which have the effect of restricting the solution space, or designer’s choice. This includes Goal (all targets) , Authority (all priority-giving parameters) and [qualifiers] for example.

## Do [PDSA] Concept \*170 January 21, 2003 B

Do [PDSA] is part of the ‘Plan Do Study Act’ Cycle (See Plan Do Study Act Cycle \*168). It 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, it is the step implementation.

Related Concepts [Do [PDSA] \*170]:

• PDSA \*168

• Plan [PDSA] \*169

• Study [PDSA] \*171

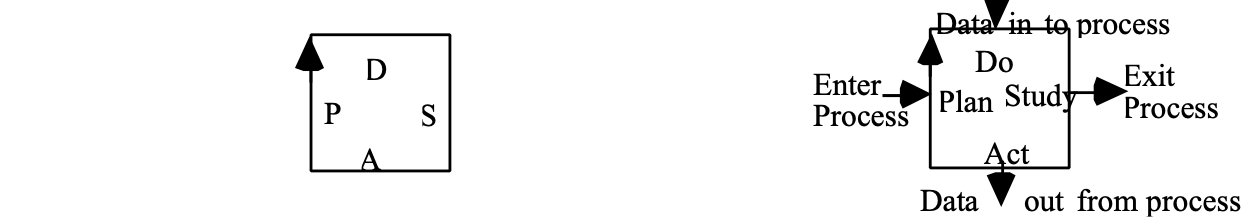
• Act [PDSA] \*278

• Evolutionary Project Management \*355

• Statistical Process Control (SPC) \*466

Keyed Icon [Do [PDSA] \*170]: There is no keyed icon specification for Do [PDSA].

Icon [Do [PDSA] \*170]: Do is the north side of the rectangular process symbol.



The four corners of the process Drawn Icon stand for the Shewhart process-cycle definition of ‘Plan Do Study Act’.

Type [Do [PDSA] \*170]: Process.

Domain [Do [PDSA] \*170]: SPC.

## Document: Concept \*180. March 13, 2003 B 23:05

A document is a readable instrument for containing and conveying ideas of any kind.

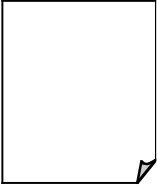
Related Concept

•  Specification\*137

Note 1. A document may be printed on paper or stored electronically and displayed on a screen or on paper. Document types include absolutely any readable information including machine-intelligible artifacts like computer source code and test scripts. A document does not have to be easily readable by unaided humans.

Note 2. All ‘documents’ are not necessarily any kind of system ‘documentation’ (specification), in the conventional (“stuff designed to describe something”) sense of the word. Nor is all ‘documentation; in the form of documents. For example a real machine, prototype, or pilot organization, can be used as ‘documentation’ of something. But this information (that machines or organizations have) is not in the form of a written document.

Icon [ \*180, Document] : standing rectangle (a standing sheet of paper), the bent paper corner in the lower right corner is a nice and desirable touch for intuitive clarity, but is optional. It is found in Microsoft Basic shapes.



Keyed Icon [ \*180, Document] : [] (made by [ & ], looks like upright page)

Tyoe [ \*180, Document] : <?>

## Documentation: Concept \*579. July 12, 2002

Documentation is written information that teaches people how to use systems.

Documentation refers to descriptions of systems for people, such as analytical documentation, instructional documentation (courseware), operational documentation.

Related Concepts [\*579 Documentation].

• specification \*137 Documentation is a class of specification, teaching people how to use a system.

• planware \*579 a class of information about human intentions (like design, requirements)

• dataware \*576 data intended for computer consumption, not human instruction

• data \*319 any signal which can be sensed, but not just those for human instruction

• infoware \*578 documentation is a class of infoware.

## Domain Concept \*374 March 5, 2003 A tg

A domain is a major field of influence or responsibility for a defined subject.

Notes:

1. Domains are declared for many reasons. These include:

• Domain classification can be used to assign responsibility for specifications of a certain domain type, to stakeholders who are interested or responsible for those domains.

• Domains can be used to understand who is the relevant authority for change or approval.

• Domains can be used to focus attention on any interesting set of ideas for learning or exposition purposes.

“A field or sphere of activity or influence” (Webster’s)

Synonyms [Domain \*374]:

• Authority Area \*374

• Competence Area \*374

Related Concepts [Domain \*374]:

• Scope \*419

• Qualifier \*124

• Authority \*005

Type [Domain \*374]: Specification Type.

## Downstream Concept \*052 November 21, 2002

Relative to an given process, ‘downstream’ refers to any processes, which are carried out later, and implies that they are impacted in some way by that upstream process.

Usage: ‘Customer use’ is downstream from ‘pilot testing’.

Rationale:

Interest in the downstream concept stems from two ideas:

1. The costs of fixing defects, which get injected upstream, become, on average, one or two orders of magnitude worse when we encounter them downstream during test or in the field.

2. The value of both ‘early quality control’ and of’ process improvement in upstream development stages’, is a function of the potential downstream damage, which apparently-small upstream defects can cause. (Compare to the ‘Butterfly Effect’ in Chaos Theory).

Parameter: Downstream. Long form = Downstream From

Keyed Icon: <~ A <~ B means A is downstream of B

Notice similarity with source arrow. Upstream is source or downstream pollution.

Example:

Goal [Processes <~ Systems Test] 99.5% “means the Goal level is for processes downstream from Systems Test.”

Type: Parameter [Relationship].

Domain: Process.

## DPP Concept \*042 November 13, 2002

*Abbreviation for Defect Prevention Process.*

## Drawn Icon Concept \*085 February 27, 2003 tg

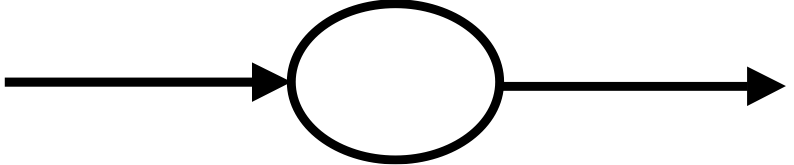
A drawn icon is a defined graphic symbol, or pictorial representation of a Planguage concept, constructed by drawing, rather than keying.

Keyed Icons are a sub-class of icons. They are constructed entirely by keying keyboard symbols.

For example:



A process



A system

Related Concepts [Drawn Icon \*085]:

• Keyed Icon \*144

• Icon \*161

• Symbol \*162

Type [Drawn Icon \*085]: Symbol.

## Due Concept \*554 March 7, 2003

‘Due’ is a specification 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 [Due \*554]:

• Deadline \*554

• Due Date \*554

Type [Due \*554]: Parameter.

Historical Note: The idea of ‘Due’ as a parameter was from an unpublished note by Jens Weber, Damlier Chrysler, Frankfurt.

## During Concept \*314 February 5, 2003

‘During’ is a specification parameter 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:

Step33: Step: {A || B During C}. “Do A, B and C concurrently.”

Step34: Step: {C [If not B||C [Step33]}. “Do step C if B was not in fact done during step C in Step33.”

The choice of using the keyed icon ‘||’ or ‘During’ is a matter of taste. Both are shown here for the sake of example.

Keyed Icon [During \*314]: || as in A||B “A is done in parallel with B.”

Type [During \*314]: Logical Operator.

## Dynamic Concept \*465 February 25, 2003

Dynamic means something happening in real time.

Antonyms [Dynamic \*465]:

• Passive

• Static

Type [Dynamic \*465]: Adjective.

## Edit [SQC] Concept \*288 November 20, 2002

The Edit sub-process of SQC primarily attempts to correct a specification by removing identified defects. The Edit process includes any defined Edit process activity, such as sending messages to other specification owners about possible problems with their specification.

Notes:

1. The specification writer (an SQC team member and also a checker themselves), usually carries out the Edit sub-process on their own. If the specification writer is not available, then someone else undertakes the editing task, and they are the editor.

2. The primary input to editing is the Editor Advice Log, which is produced by the SQC team during the Specification Meeting. Secondary inputs are all documents, which participated in the SQC Checking process, and any notes (‘scratchings’), which checkers made but chose not to formally log.

3. The editor reads the issues in the Editor Advice Log and makes corrections wherever they find and agree that there are defects requiring correction in the main specification. They can choose to correct far more defects than were reported, based on the stimulus of the sample of defects presented to them. They can also choose to not fix things, which they judge are not major defects. They should somehow try to bring the specification to an exit-able defect level.

4. Where the authors or process owners of any other documents involved in the SQC are concerned, other reasonable action has to be taken, such as sending out change requests. The editor is at liberty to search (and take appropriate action) for other defects, not reported, anywhere in the specification set.

5. The Edit process is reviewed, and ultimately approved, by the team leader in an SQC sub-process called ‘Edit Audit.’ More detail can be found in [GILB93] (See ‘Editing’ and ‘Follow up’).

Type: Process.

Domain: SQC.

## Edit Audit [SQC] Concept \*289 November 20, 2002

Edit Audit is an SQC sub-process, carried out by a team leader, to verify that reasonable and complete editing has been done.

Note: Edit Audit will try to make sure that valid main specification rules, the editor advice log, relevant editing and specification procedures, and main specification sources have been reasonably heeded when editing the specification.

Synonym: ‘Follow up’.

Related Concepts:

• Edit [SQC] (\*288)

• SQC (\*051)

Type: Process.

## Editor Concept \*277 November 20, 2002

During the Edit sub-process of SQC, an ‘editor’ is a person who evaluates issues (and other Editor Advice Log information) and takes appropriate action on them.

Note: The appropriate action might include correcting something if the issues are defects, or sending change requests to other document owners if action needs to be taken at that level.

Type: Role.

Domain: SQC.

## Editor Advice Log Concept \*296 November 20, 2002

An Editor Advice Log is written formally by a scribe at an SQC Specification Meeting. It is based on oral reports from checkers.

Notes:

An Editor Advice Log contains three types of items: issues, improvement suggestions, and questions of intent for the specification writer. The log is written with no debate, discussion or consensus, irrespective of other teammates’ opinions. It is written in a ‘brainstorming’ mode (volume now, analysis later).

The log is the primary input to the Edit sub-process, where the editor (usually the writer and owner of the specification) seriously evaluates each item. The editor makes all possible corrections to their specification, according to best available rules and sources, in all parts of the specification, not just the sample or chunk reported on during the Specification Meeting.

An Editor Advice Log is only written when the Specification Meeting entry process decides it is worthwhile. It would not be worthwhile if too many or too few major defects were found in checking. Nor would it be worthwhile if the checking process had not been done properly, such as at ‘too fast a checking rate.’ In cases, where ‘no logging’ is chosen, there is the option of giving the specification writer/editor the ‘scratchings’ (informal notes made by checkers during their checking work). This is a practical way of giving the specification writer /editor some advice without taking any time for writing a formal log. There is even an additional option of doing a partial log as a sample of issues for the writer/editor, without necessarily logging all the discovered issues. This can then be supplemented by the ‘scratchings’.

Synonyms: Issue Log, Author Advice Log.

Type: process output specification.

Domain: SQC.

## Effect Concept \*441 July 26, 2001

The result (‘the effect’) of using a system performance attribute (the ‘cause’) .

Formally:

• the differential value to a defined stakeholder,

• caused by a defined level of a defined system attribute,

• acting on a defined stakeholder,

• in a defined stakeholder environment.

More simply and less formally: an effect is the stakeholder joy from a system performance attribute.

Notes:

Note 1. The effect is the result of using a system performance attribute created in one system, as input to another system or process. It is an output or performance attribute of that second system or process.

Instances:

• how does 99.98% availability effect a stakeholder who has been accustomed to 95% availability, and is quite satisfied with that level for their purposes. The stakeholder would not pay anything more for an increase beyond 95% availability?

• in a second case, another stakeholder with the same old and new system situation as above, but with different ‘customers’ might be losing 5% of their business because of the lack of availability, and would be willing to pay what it was worth, in increased sales ability, to get higher availability.

(S1)------P<-------> Q Q ----------|P-------->(S2)------e1<------😐E1<-----😐E2<----->PL

A level of performance P is produced for quality q in system S1. This performance level (P) is used in system S2 and results for some stakeholder (1) in the effect E1 (like more sales) and for another stakeholder (2) in a different effect ‘E2’. The effect could be stated in direct terms (€ Sales) or relative to some stakeholder benchmark like % increase in sales over e1) Keyed Icon: any benchmark or generic level of performance. For example O----|---->, or O-----<-----<<----->Source: of concept Kai Gilb, End 2000

## Effectiveness Concept \*053 April 26, 2003

Effectiveness is a measure of how good the current attributes of a system are at delivering the stakeholders’ current demands of the system.

Effectiveness is a raw measure of impact, and does not contain information about the costs of making that impact (performance *in relation to cost* is the role of an *efficiency* measure).

In ISO 9000:2000 ...

Effectiveness - extent to which planned activities

are realized and planned results achieved.

Efficiency - relationship between the result achieved

and the resources used.

Notes:

1. Several ‘measures of effectiveness’ can be defined. Actual achievement of benefits compared to expected benefits is one possible measure.

Related Concepts [Effectiveness \*053]:

• Performance \*434

• Measure of Effectiveness \*486: See also [SPROLES02].

• Benefit \*009

• Measure of Performance \*482: A measure of performance (MOP) considers only the effectiveness of a set of performance attributes: A Sum of Performance for a delivered Evo Step can be used as such a measure. See also [SPROLES02].

• Sum of Performance \*008

• Efficiency \*054

Type [Effectiveness \*053]: Metric.

## Efficiency Concept \*054 April 28, 2003 tg

Efficiency is ‘effectiveness’ in relation to cost.

Example:

The defect detection efficiency of the Specification Quality Control (SQC) process is about ‘one major defect found and fixed’ per ‘work-hour invested’. This compares well to the downstream testing and rework processes (debugging), which are an order-of-magnitude less efficient, and even more, the downstream field rework costs, which are two orders of magnitude less efficient.

Sources: [GILB93, WHEELER96].

Notes:

1. A benefit to cost ratio or a performance to cost ratio would be a measure of efficiency.

Synonyms [Efficiency \*054]:

• Cost Effectiveness \*054Related Concepts [Efficiency \*054]:

• Effectiveness \*053

• Performance to Cost Ratio \*010

Type [Efficiency \*054]: Metric.

## Efficiency (noun adjective): Concept \*394 April 1, 2003

**The noun adjective ‘Efficiency’ can be used to refer to any set of Performance to Cost Attribute ratios.** For example: ‘Efficiency Attributes, Efficiency Requirements, Efficiency Benchmarks, Efficiency Targets, and Efficiency Records.

Type [Efficiency (noun adjective) \*394]: noun adjective, metric concept.

## Effort: Concept \*368. July 26, 2001

Human-energy task cost.

The human resources needed to accomplish something.

Related concepts:

• Resources (which is much broader in the set of possible resources than Effort. Effort is intentionally narrowed to human effort).,

• Engineering Hours (a common measure of effort).,

• Work hours (a measure of effort, and a politically correct form of the traditional ‘man hours’)

Type: quantification concept.

Scale:

Effort: Scale of measure: the defined [type of effort, default = engineering hours] needed to accomplish a defined [task].

Usage Example:*”HP does Platform Development ‘Product Portfolio Planning’ by defining a ‘target effort savings’, meaning “the platform will reduce product investment by X engineering months”*

*[JANDOUREK96, page 61]*

*Keyed icon: the resource icon with any useful benchmark and target icons. For example:*

------p<------>>m---]c--->O

p = Past level of the effort, m = Fail limit of the effort, c = a limit constraint,

------>O is the resource scale, defined in terms of human effort.

In ISO 9000:2000 ...

Efficiency - relationship between the result achieved

and the resources used.

## Element Concept \*022 February 5, 2003

An element is any identifiable individual part of any set.

Note: An element can itself be composed of other elements. It can be complex: element does not imply elementary (See Elementary \*055).

.

Synonyms: \*022

• Component \*022

• Is Part of \*022 (parameter)

Type: structure descriptor.*Note 1. An element can itself be composed of other elements. It can be a Complex element.*

*Element does not imply Elementary (\*055).*

*Keyed Icon: \*022*

*X {y}. meaning y is included in X*

Related Concepts \*022

• Consists Of \*616

• Includes \*391 Includes can give a partial list of contents.

Type: Specification Object

## Elementary Concept \*055 February 1, 2003, 31 August 2014

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 sub-divide a compound 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 [Elementary \*055]:

• Compound \*023

Type [Elementary \*055]: Adjective.

## Ends: Concept \*316. August 17, 2001tg 2013

‘Ends’ is a Planguage parameter that specifies a condition that a defined term has completed successfully (is true).

Type: Planguage parameter.

Usage: to indicate sequential priority in planning time consuming events such as Evo steps.

Related Concepts:

• Starts **315 Icon ‘**\*’

• Before \*312 Icon =>

• After \*313 Icon =>

• In Process \*113 ^

Examples:

Peace [X Starts Before A Ends] … meaning X must be started before A ends or completes successfully.

Peace [X \* => A •] same as above iconically

A\* A Starts

A• A Ends

A^ A is in Process (after \* and before •)

A^ => B• A is in Process Before B Ends

Step23: XX [XX After A Ends] ... meaning XX will only start when A is completed.

Keyed icon [Ends]: ‘•’

Examples of ‘·’ (Icon for Ends)

***X [X => A***·***]***

Meaning X must be started before (=>) A ends or completes successfully (•).

***Step23: XX [A***·***]***

*Meaning XX is only to be planned to start when A is completed [A•]. ‘A Ends’ (A•) is a condition for XX to be ‘true’ as a step plan.*

It is implicit in Evo planning that reference to a function, or a design idea, specification implies the *implementation* of that function or design idea. It is obvious that a complete and correct implementation of that specification is a condition for ‘Ends’.

## End: Concept \*100. July 26, 2001

Future requirement or objective. Desired end state.

Generic synonym for \*100.Objective.

Related Concepts: Means, \*047 (the way to reach the ends).

@ Keyed Icon (same as Objective @, \*100).

Note for the more-specific concept of Goal or Target, use the set of target icons like

------|--->>--->---->.

“Think well to the end”

“Consider first the end”

Quoted in Gelb: How to Think Like Leonardo da Vinci, p. 243

## Engineering Concept \*224 June 28, 2003

Engineering is

• an Evolutionary Process, • using practical Principles, • in order to determine,• and identify the Means to deliver, • the best achievable Performance and Cost levels balance, • for optimal Stakeholder satisfaction, • in a complex risk-filled environment.

"The Engineering Method is the use of heuristics to cause the best change in a poorly understood situation within the available resources""Engineering is a risk-taking activity. To control these risks, engineers have many heuristics:1. They make only small changes in what has worked in the past,

2. They try to arrange matters so that, if they are wrong, they can retreat, and3. They feed back past results in order to improve future performance.""Engineers cannot simply work their way down a list of steps, ... but ...they must circulate freely within the proposed plan ..."

Prof. Billy V. Koen, Univ. of Austin, Texas [KOEN84, KOEN03]

Notes:

1. Here is my rephrased version of Koen's definition of engineering:

(It is a variation on my definition above)."Engineering is the use of principles in a systematic way, to find designs, that deliver the value requirements, within resource constraints, under conditions of uncertainty".*Some Comments on this Definition*: • 'Principles' are rules of thumb that guide the choices and actions of engineers. Principles are not rigid laws. They reflect the experience learned from previous engineering projects. I see professional codes of conduct playing a part as well.• 'In a systematic way' must include brainstorming, and testing random insights.• 'Deliver' implies a process that continues beyond design stages to project delivery stages. Success is determined by successfully meeting all the specified and agreed requirements. • 'Conditions of uncertainty' exist because we must deal with combinations of designs, being applied into ever-changing environments. The results are 'complex to predict' and the resulting risk has to be handled.

2. I want to make it clear that whenever I use the term engineering, I mean to imply the use of quantification and measurement as a basic tool, rather than more intuitive methods.

Related Concepts [Engineering \*224]:

• Systems Engineering \*223

Type [Engineering \*224]: Process.

## Entity Concept \*645 January 7, 2004

Entities exist. An entity is a real thing.

An entity is real in the sense that it has provable existence, and it has performance and cost attributes. An entity can be natural or human made. It can be conceptual or physical.

Notes:

1. an entity is not necessarily a physical object. It can be a policy, a law, a contractual condition, a motivator. These are still real for people. They have performance and cost attributes.
2. designers choose entities because of their attributes. The description of the entity is called a design specification or an architectural specification, usually shortened to a design or the architecture.
3. but if all specifications or descriptions of entities were burned, the entities themselves would continue to exist.
4. I am not sure if all entities have a function, but it seems that they do. The function is what the entity does.
5. The distinction between an entity and a system is that entities must have real existence and systems can be imagined or conceptual in nature.

Synonyms [Entity \*645]:

• thing

•

Related Concepts [Entity \*645]:

• Attribute \*003

• System [Planguage] \*145

• Function \*069

• Design \*047

Type [Entity \*645]: Systems Engineering Concept

## Entry Condition Concept \*056 February 22, 2003 tg

An entry condition is a written part of a work-process standard. Entry conditions are usually found in sets. The relevant set of entry conditions is evaluated in an entry process. They are used to determine if there is entry permission to a task process.

Notes:

1. Entry conditions are used to decide if it is economic and safe to enter the main task of a process. They are designed to:

• prevent entry into a task process if it would be failure-prone

• cause us to remove obstacles to successful task process execution

• provide a ‘standard,’ which we can continuously update whenever we get experience or knowledge to improve our productivity.

2. All the entry conditions must normally be met for permission to enter into the main process task.

Example:

Entry Condition 1: The requirement specification shall have exited from SQC with a maximum estimated remaining major defects/page of less than one.

3. For any work process, there are usually several entry conditions, which should apply. They can be grouped in sets into ‘generic’ and ‘specific’ entry conditions.

4. Entry conditions are one type of work process standard. They are one means of ‘process improvement’ since they provide a mechanism for ‘organizational learning’. (Any experiences of conditions that warn of impending problems can be noted in the form of entry conditions. They are then available to anyone in an organization, even those people, who have not personally had the ‘bad’ experience, which may have led to the establishment of the entry condition.)

5. Note: Nov 1 2003 Gate is a generic word for both Entry and Exit conditions. <-McConnellType [Entry Condition \*056]: Condition [Process].

## Entry Process Concept \*057 February 22, 2003tgAn entry process involves evaluating any applicable entry conditions for a process. The process itself is defined by a main task. When *all* entry conditions are met, the main process task may officially commence.

Notes:1. The purpose of an entry process is to help to avoid failed (or uneconomic or time wasting) execution of the main task. We need to work to remove any entry condition failures. Otherwise, waiving or ignoring failed entry conditions is a possibility, with determinable risks.

2. When referring to the SQC Entry Process (or any other specific named Entry Process), capitals may be used, to distinguish it from the basic ‘entry’ concept, which applies to all processes.

Abbreviation [Entry Process \*057]: ‘E’ <- IBM convention originated by Ron Radice/Watts Humphrey. (Abbreviation for a process control cycle is ETVX or ETX: ‘Entry Task Validation eXit’.)Keyed Icon [Entry Process \*057]: ➔[…]

Example:

➔[Design Process]

Type [Entry Process \*057]: Process.

## Environment Constraint: Concept \*491 Oct 27 2001 2107 London

A Environment constraint is any system constraint that is not a focus system constraint.

Environment constraints are particularly intended to describe constraints on the life cycle of a product or system, such as:

• design and architecture phases

• construction and acquisition phases

• implementation phases

• operational use phases

• shut down phases

Constraints are of four basic types:

• function constraints

• performance constraints,

•  resource constraints and

• Environment constraints.

Environment constraints can be expressed as binary constraints or as scalar constraints.

Synonym: non-focus system constraint, environment system constraint.

Related Concepts:

• Environment system \*493

• Focus system \*492

• Binary Constraint \*467. A constraint which is formulated in terms which are binary (true/untrue) in nature.). These are of two types veto and demand.

• Demand Constraint \*461

• Veto Constraint \*462

• Constraint \*218

• Scalar Constraint \*468

• Performance Constraint \*438

• Resource Constraint \*478

• function constraints \*469

Type: Constraint

## Environment System: \*493 August 17, 2003

An environment system is any system related to a defined Focus System.

For example it could be a development process or development tool for a product (designated as the defined focus system).

Synonyms [Environment System \*493]:

• Enabling Systems <-MIL-STD-499B

Related Concepts [Environment System \*493]:

• Focus System \*492

• Environment Constraint \*491

Type:???

## Error Concept \*274 December 9, 2004An ‘error’ is something done incorrectly by a human being.

Error/Slip 🡪 (Specification Issue) 🡪 Specification Defect 🡪 Fault/Bug/System Defect 🡪 Malfunction

Notes:

1. Errors are usually committed unintentionally; they are often forced to happen by ‘bad’ work processes (Statistical Process Control Theory [DEMING86, JURAN64 &74]).

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.

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 (A.D. 46-120) “For to err in opinion, though it be not part of wise men, is at least human.”*Synonyms [Error \*274]:

• Slip \*274

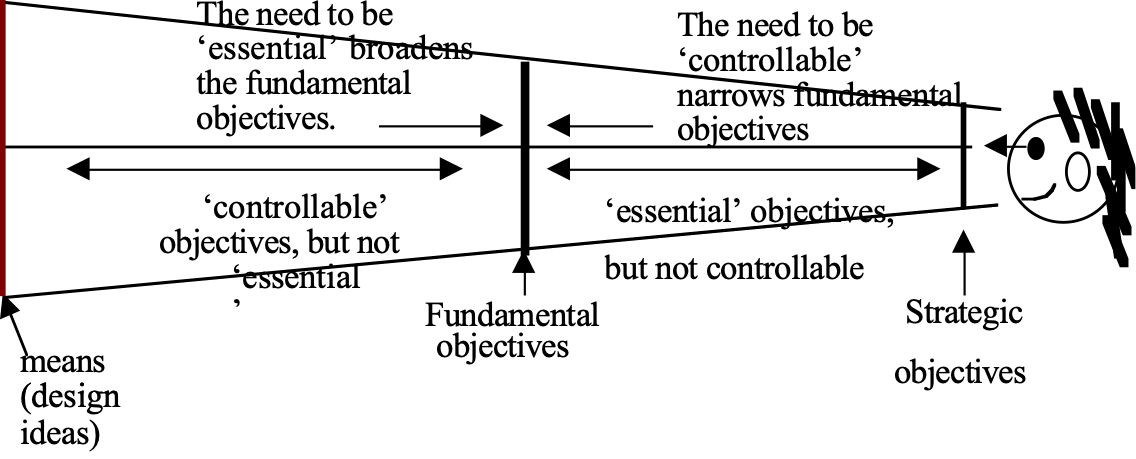
Related Concepts [Error \*274]:

Defect: Specification Defect \*043

Fault: Bug: System Defect \*339

Malfunction \*275

## Essential Objective "essential" : Concept \*231. January 8, 2003

Essential objectives are the ones which are capable of influencing the Fundamental objective when they are reached or implemented "fundamental objective" (See Keeney). Compare with ‘Controllable’.

The effect of essential and controllable properties on the identification of fundamental objectives [after basic ideas of KEENEY92, p.84].

## Estimate Concept \*058 January 8, 2005, 21 June 2011 (‘derived’)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 derived 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.

Synonym [Estimate \*058] predict.

Keyed Icon [Estimate \*058]: # “The same icon as Cell. This icon is symbolic of an Impact Estimation table. In the USA, this icon is the ‘number’ symbol.”

Parameter [\*058]: Estimate: or ‘Estimated:’

This is a Benchmark specification used when there is no past or trend or record, and you need to develop an estimate for a certain period ( for example ‘ This Year’)

Revenue:

Ambition: exceed USD 30 million

Scale: US$ gross revenue in a single financial year.

Past [2003] $? Mill.

Estimate [2004] $? Mill.± 10% <- CFO

Goal [2006, Constraint = Operating Margin < 20%] $30 mill. <- CEO?

Type [Estimate \*058]: Metric, Parameter.

## Estimate, To Concept \*059 January 8, 2005

In Planguage, to ‘estimate’ is:

• the process of arriving at a judgement,

• 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”

Source: Webster’s New World Dictionary.

Estimation is not to be confused with Quantification or Measurement.

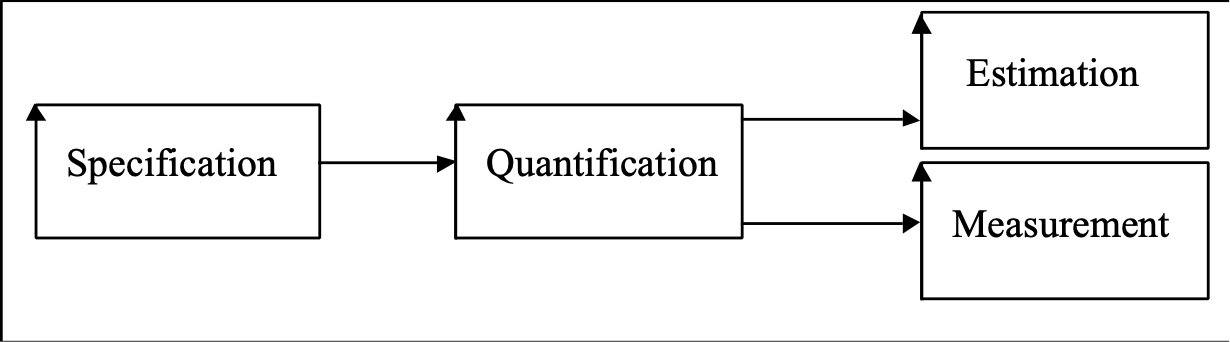


Figure \*059: The relationship between four engineering process concepts.

Related Concepts [Estimate, To \*059]:

• Estimate \*058 (noun, an estimate)

• Quantify \*386

• Specify \*239

• Measure \*386

Synonym [Estimate, To \*059]

• Predict, To (Verb)

Type [Estimate, To \*059]: Verb.

## Estimated Remaining Defects "estimated remaining defects" : Concept \*060. July 26, 2001

This is an estimate of defect density remaining in a specification at any given time.

Usage 1. The estimate is based on

• defects found (using knowledge of % defect-finding effectiveness) and

• defects attempted removed (using knowledge of % effectiveness of the correction process).

Usage 2: The defect density is usually expressed in Major defects per Logical Page.

*EXAMPLE OF ESTIMATING REMAINING DEFECTSFor example,*

*• if the defect detection process had a historic and stable effectiveness of 80%, and*

*• you found 80 Major defects, then*

*• you might estimate that there are a total of 100 Majors,*

*•  of which you found 80 and 20 Major defects you did not find.*

*• If you correct the 80 you found then*

*• 20 not found remain (plus and minus some uncertainty factor of about 30% of the total remaining).*

*• If you historically fail to correct 25% of what you try to correct, then*

*• to the 20 you did not find or fix*

*• you need to add 20 (25% of 80 correction attempts) you failed to correct correctly.*

*• This gives a total estimate of 40 Major defects remaining (20 never found and 20 not fixed correctly),*

*• after you find 80 and try to correct them. I have found that this method works well regularly, and can be confirmed by an SQC process that manages to find a predicted number of defects (for example 80% of the 40 remaining in the example above, i.e. about 24 Majors), at various clients. It gives the right order of magnitude, even when there is little history, and conditions are somewhat unstable. If this seems too complicated, then use the following rule of thumb, which will make you ‘humble’.*

For every defect you find and fix there remain that many more in the system.

*If that doesn’t work for you, try Gerald M. Weinberg’s Principle, (Psychology of Computer Programming)*

“There is always one more bug remaining”(Even after you think you have removed the last bug).

## Evaluation Effort [Specification Defect Density] Concept \*061 November 21, 2002

This is the cost (in work hours) of determining the quality level (Remaining major defects per page) of a specification.

Notes:

1. The process to do the evaluation is assumed to be SQC, but other processes to determine the quality level could be used.

2. Use of sampling will dramatically reduce the evaluation cost, time and effort, at the expense of a mild decrease in accuracy and credibility.

Rationale: The reasons for collecting data on the cost of SQC are as follows:

• to ensure that SQC is cost-effective; and that the savings sufficiently outweigh the

evaluation costs

• to reduce the costs to a minimum, and

• to motivate us to use inexpensive methods such as sampling.Type: Metric. Domain: SQC.

## Event Concept \*062 May 27, 2003

An event is a specified occurrence.

Examples:

• 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.

Examples:

First Sale: Event: We make our first sale of refrigerators to the USA.

Goal [First Sale]: 99% Or Better.

Past [Last Year, Europe, First Flight]: 98%.

First Flight: Type: Event. Description: Successful first flight <officially logged>.

6. To attempt a more detailed definition[[3]](#footnote-3): 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 co-ordinates.

Synonyms [Event \*062]:

• Happening

• Occurrence

• Point in space-time

Related Concepts [Event \*062]:

• Qualifier \*124

• Condition \*024

• Status \*174

Type [Event \*062]: Parameter, Scope Concept.

## Evidence Concept \*063 January 20, 2003

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.

Examples:

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.

Record [GB, 2000]: 60 Hours? 🡨 R. Branson, ‘Losing My Virginity’.

#.🡨 Branson claims to have beaten previous record of 59 hours slightly.

‘#.🡨’ gives the evidence for the uncertain (?) estimate of 60 Hours.

Keyed Icon: *#<-*  “source (<-) of the estimate (#).”

Type: Parameter, Systems Engineering concept.

Domain: Impact Estimation.

## Evo Concept \*355 June 5, 2003

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.*

## Evo Plan Concept \*322 May 26, 2003

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.

3. An individual Evo step can be ‘rolled out’ across a system. That is, the design ideas making up the Evo step are developed,. Then production and delivery is repeated over several Evo steps; with delivery to different parts of the system for each step. In such cases, the Evo plan is likely to be the place where the details of the impacts of the repetition of step delivery are held.

Synonyms [Evo Plan \*322]:

• Evolutionary Plan \*322

• Evolutionary Delivery Plan •322

Related Concepts [Evo Plan \*322]:

• Evolutionary Project Management (Evo) \*355

• Evolutionary \*196

Type [Evo Plan \*322]: Specification Concept.

## Evo Step Concept \*141 May 11, 2003

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 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.



Figure \*141: Evo Step packaging. Delivery-specified Evo Steps: same underlying step specification, but two different times and places.

Synonyms [Evo Step \*141]:

• Step \*141

• Evolutionary Step \*141

• Build, Increment, Installment, Release: Near synonyms depending on whether there is use of feedback and dynamic change [ROYCE98]

Related Concepts [Evo Step \*141]:

• 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

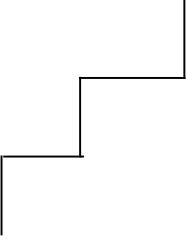
Example:

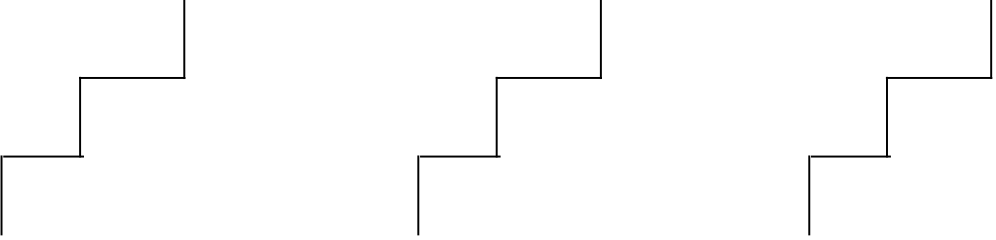
S23: Step [<Time, Place, Event to be determined>]: F1, F2 [Europe], D3 [China].

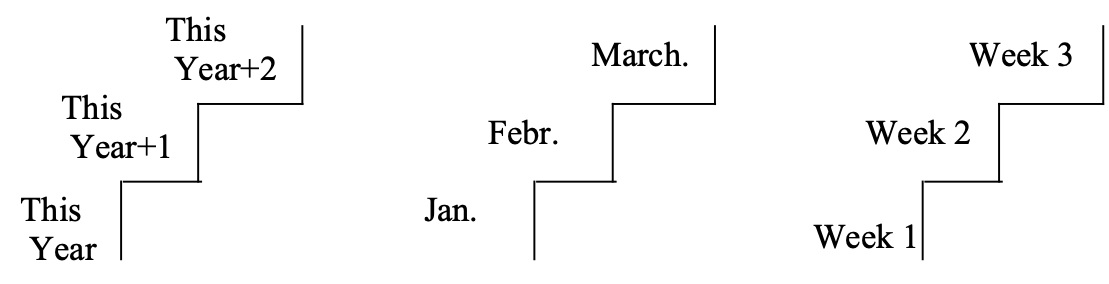
The main difference between an Evo step package (above example with undetermined qualifier) and a delivery-specified Evo step is that the latter has been assigned a sequence or timing and a place of application qualifier. The Evo step package is just a specification of the step contents. An Evo step 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.

Keyed Icon [Step \*141]: ->☺ or ->:) “Symbolizing ‘Impact on stakeholder’.”

Note: The ☺ symbol is sometimes automatic ‘correction’ for the colon and right parenthesis keyed symbols, in Microsoft Word.

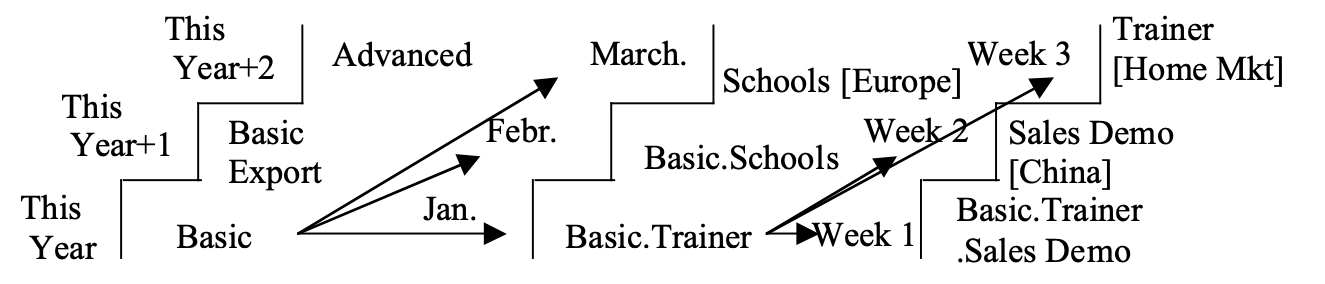
 This above drawn icon is a series of any number of steps, each one representing an Evo step. I have used this for decades as a graphic icon for a set of evo steps.

A hierarchical set of evo steps (above). The left most can for example represent ‘years’, the middle one ‘months’ or ‘quarters’, and the rightmost steps ‘weeks’ or elementary steps.



[Editor Note to Publisher: Please correct Months on redrawing. Thanks]

And we can add specification in the form of tags of function, design and qualifiers.



Add lines or arrows to indicate step and sub-step relationships.

Type [Evo Step \*141]: Specification Object, Parameter.

## Evo Step Specification Concept \*370 May 29, 2003

An Evo step specification is the formal written definition of the content of an Evo step .

Notes:

1. Outline Evo step specifications are produced towards the end of the initial design engineering process (that is, prior to evolutionary project management (Evo) commencing) as part of producing an Evo plan. The main concern at this stage is that there is sufficient information to enable the next step to be selected from the potential candidates.

During Evo, additional outline step specifications will be written as additional step possibilities are identified.

2. Detailed Evo step specification occurs within Evo, once the step has been selected as being the next for implementation.

3. An Evo step specification can differ from the reality of the delivered step content.

Template [Evo Step Specification \*370]: Evo Step Specification.

Synonyms [Evo Step Specification \*370]:

• Step Specification \*370

• Evolutionary Step Specification \*370

Related Concepts [Evo Step Specification \*370]:

• Design Engineering \*501

• Evolutionary Project Management (Evo) \*355

• Evo Step \*141

Type [Evo Step Specification \*370]: Design Component.

## Evolutionary Concept \*196 March 20, 2003

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

Abbreviation [Evolutionary \*196]:

• Evo \*196: *Readers will have to bear with me that I use this abbreviation for both ‘evolutionary’ and ‘Evolutionary Project Management.’ The underlying concept is the same <- TG.*

Related Concepts [Evolutionary \*196]:

• Evolutionary Project Management \*355

• Evo Plan \*322

• Evo Step \*141

• Plan Do Study Act Cycle \*168

Type [Evolutionary \*196]: Adjective.

## Evolutionary Project Management Concept \*355 April 14 2005 EIN

A project management process delivering ‘high-value-first’ progress towards the currently *defined and approved requirements*, and then seeking to obtain, and use, realistic, early feedback.

Key components include:

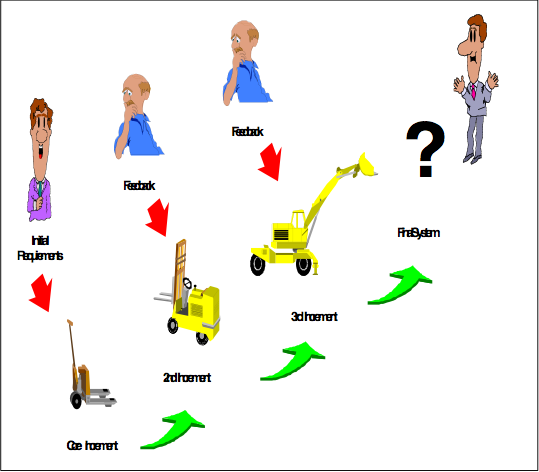
• frequent delivery of system changes (steps)• steps delivered to stakeholders for real use• feedback obtained from stakeholders to determine *next* step(s)• the existing system is used as the initial system base• small steps (ideally between 2%-5% of total project financial cost and time)• steps with 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)

Figure \*355: Conceptual view of Evo [HUMMEL02]. Notice that the design changes over time: it does not just get ‘incremented’. Of course, it is not a requirement that the design has to fundamentally change at each step! Compare with Figure \*318 in Incremental Development \*318.

[Editor note to Publisher: Please replace ‘Core Increment’, ‘2nd Increment’, ‘3rd Increment’, ‘Final System’ with ‘Step 1’, ‘Step 2’, ‘Step 3’, ‘Step 4’. Thanks]

Abbreviations [Evolutionary Project Management \*355]:

• Evo \*355

Synonyms [Evolutionary Project Management \*355]:

• Evo Management \*355

• Evolutionary Delivery Management \*355

• Rapid Delivery Management (Acronym: RDM), Result Delivery (These synonyms are used within Jet Propulsion Labs. [SPUCK93]

• Synch-and-stabilize or Milestone Approach (These synonyms are used within Microsoft [CUSUMANO95].

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.

Type [Evolutionary Project Management \*355]: Method.

*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 [MILLS80]. [LARMAN03] gives a comprehensive history of the method.*

.

## Except Concept \*389 July 12, 2003

‘Except’ is a parameter, which means that the following term or expression is an exception from the previous term or expression.

Example:

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

Keyed Icon [Except \*389]: -

Type [Except \*389]: Logical Operator.

## Exit [SQC] Concept \*290 November 20, 2002

Exit is the name of an SQC sub-process. The SQC team leader checks a set of defined SQC exit conditions, and decides whether to allow the specification to exit, based on the exit condition status; or whether to take other action, as a result of unsatisfied exit conditions.

Note: The maximum recommended level of estimated remaining major defects per page in the specification – an SQC measurement - is an important exit condition; but not the only one usually.

Synonym: SQC Exit.

Related Concepts:

• Exit Condition (\*064)

• Exit Process (\*065)

• Entry Condition (\*056)

• Entry Process (\*057)

• Team Leader \*281

Type: Process.

Domain: SQC.

## Exit Condition Concept \*064 February 22, 2003

An exit condition is a written part of a work-process standard. Exit conditions are usually found in sets. The relevant set of exit conditions is evaluated in an exit process. They are used to determine if there is exit permission from a task process. All the generic and specific exit conditions must normally be met, in order to exit from a process, thus releasing work products to the other ‘downstream’ processes.

Examples:

Exit 33: The remaining quantity of major defects (potentially costly rule violations) in a document may be calculated (based on defects found and fixed) to determine if it is wise to release (exit) an engineering or management specification.

We can decide this by considering the alternative cost consequences (compared to exit ‘now’) of the remaining defects, in later test and field use phases.

At stages later than specification, major defects are one and two orders of magnitude more expensive to deal with.

***Exit 13: The system design must arguably have a complete set of designs to reach all performance targets, within resource targets and generic constraints, considering acceptable risk (negative uncertainty), credibility and safety margin.***

Related Concepts [Exit Condition \*064]:

• Entry Condition \*056

• Exit Process \*065

Type [Exit Condition \*064]: Condition [Process].

## Exit Process Concept \*065 February 21, 2003

An exit process involves evaluating any applicable exit conditions for a process. Only when *all* exit conditions are met, can the main process task officially terminate and any process output be released to the next process.

Notes:

1. If any conditions fail, the exit process includes work to improve exit condition status to the required level. If all conditions are OK, then a process product can be released to its customer, and the process is officially completed. Abbreviation [Exit Process \*065]: X “As in ‘eXit.’”

Related Concepts [Exit Process \*065]:

• Exit Condition \*064

• Entry Process \*057

Keyed Icon [Exit Process \*065]: [ ]➔ or ^[ ]->

Examples:

[Requirements]➔ Maximum 1 estimated remaining major defect/page.[Design]➔ Achieved Jan 23rd.

^[Requirements] “This is a reference to the Requirements Process alone, not the exit.”

➔[Test Planning] “This is entry to test planning reference.”

[Installation]➔ “This is a reference to the Installation process Exit”

Goal [IF [Acceptance Test]➔, Initial Delivery]: 60% “The condition is that the Acceptance Test has Exited.”

Type [Exit Process \*065]: Process.

## Expert Concept \*678. December 2 2014

A source that knows the state of the art levels in a defined discipline, and how to make them happen in practice.

Notes:

1. Source, is used to include the idea that ‘it does not matter if the expertise comes from a person, a group, or a written source such as a book, a paper, presentation, video, or internet’.
2. State of the art: is the border between what we in a given domain, have actually achieved in reality, and beyond that border, where ‘no man has gone before us’. We ‘know we don’t know’ how to go beyond that border. So the expert knows what is ‘possible, for sure’; and which might therefore be ‘impossible for now’, costly, risky, or challenging.
3. Knowing ‘how to make it happen’, means the source knows the necessary architecture, design, strategy, or means to achieve the state-of-the-art levels in practice. We argue that merely knowing the existing ‘records’ is one simple level of expertise. But is almost without real value if we cannot know **how** they were achieved in practice. Because we will have difficulty replicating or beating the feat.
4. There is an implication that the ‘source’ also has **knowledge of** the cost aspects for example (capital, operational, time) of the means to achieve the state-of-the-art levels, and probably also knowledge about cheaper options at lower levels of performance.

**Expert** is a Planguage parameter for specification which indicates the subject matter best available expert person or instance.

Thanks to R[aul.giurgiu@gmail.com](mailto:aul.giurgiu@gmail.com), Oslo for inviting me to discuss and define this concept, 1 Dec. 2014.

## Explicit Parameters: Concept \*378. July 27, 2001

Parameters in a specification that are defined by *direct use* of their defined parameter name.

For example:

Scale: Probability of [Fault].Goal [Fault = Show Stopper]. “all 3 terms {Scale, Goal, Fault} are Explicit Parameters’

The ‘Scale’ is an explicit generic Planguage parameter. ‘Fault’ is another explicit parameter, project defined (see \*377 Generic Term for example of definition), for the generic term ‘Fault’ (identified in the ‘Scale’ specification).

*The alternatives to ‘explicit parameter use’ would be:*

*• defining a parameter by ‘default’ (Default Value \*447) or*

*•  using ‘sequence of definition’ (1st, 2nd, 3rd), see* Scale Qualifier: Concept \*381 and Scale Variable \*446*, or*

*• simply by recognizing the parameter as having a unique definition (‘Show Stopper’, example above, defined once) and belonging to a particular generic term (like ‘Fault’).*

*Explicit parameters are a device for making specification more obvious to the reader.*

## Explosion "explosion" : Concept \*066. July 27, 2001

Explosion is a ‘dramatic’ word for ‘hierarchical decomposition of complex specifications into useful, more elementary, detail’.

Antonyms: implosion, synthesis (Descartes), aggregating

Synonyms: decomposition, detailing, breaking down into sub-components, elementing (concocted for fun July 27, 2001 Tg)

Also known as Cartesian Analysis (René Descartes, French Philosopher).

## Expression Concept \*302 June 1, 2003

A Planguage ‘expression’ is part or all of a Planguage ‘statement’: it is one or more related Planguage components of any kind.

Expressions are defined by one or more Planguage terms.

Expressions are comprised of Planguage language concepts. These concepts include: {parameters, icons, tags, qualifiers, sources, fuzzy terms}.

For example (below), a ‘term’ (in this case a ‘tag’), ‘Adaptability’ is described by a set of three ‘statements’. Each statement, using one or more ‘expressions’ is a ‘definition’ of the tag. The ‘expressions’ themselves are composed of ‘terms’.

Adaptability:Ambition: Ability to make changes quickly and cheaply from Customer’s point of view.Scale: Calendar days needed from <customer request> to <successful first delivery>.

Next Year: Goal [End of Next Year]: 7 days “Half of our present adaptability time” <- Sales Plan.

‘Adaptability’ itself, once defined, can be reused as a term in another expression (below).

MTBF: Goal [Adaptability, Next Year]: 6,000 hours.

Related Concepts [Expression \*302]:

• Statement \*140: See diagram showing how Expression relates to Statement.

• Term \*151

• Definition \*044: A definition can comprise one or more statements.

Keyed Icon [Expression \*302]: ‘ ’ “Single quotes can be used round a set of terms to identify them as forming a single expression.”

Type [Expression \*302]: Specification Concept.

## Expression Quotes: Concept \*329. July 27, 2001

Expression quotes are ‘single quotes’ at the beginning and end of an expression. They are used to make it clear that this is a *single* expression, using a set of related terms.

*Rationale: They are useful when the expression is complicated and some doubt could arise as to the expression.*

*Note, these are quite distinct from the “double quotes” used to place comments anywhere in Planguage, and to express Data.*

Goal [‘Time = After 2002 [Europe, IF Peace]’, GB] 60% “these are double quotes”.

Usage:

An alternative way of doing this is to define the complex expression:

Peacetime: Time = After 2002 [Europe, IF Peace]

And use it through its tag, instead of an expression in single quotes:

Goal: [Peacetime, GB] 60% .

## External Concept \*497 May 9, 2003

When describing the relationship amongst objects, external is an adjective, which applies to any object outside the scope of a given object.

Examples:

• External stakeholders of a system

• The influences of the external environment on a system

Notes:

1. ‘External’ is used with regards the ‘place’ scope of an object, rather than the ‘time’ or ‘event’ scope. ‘Where’ does an object stand with regard another object: within its extent of influence (internal) or outside it (external)?

2. If an object is external (outside the scope) of an object, it means it is not under the control in any way of the given object. This does not mean that no influence whatsoever can be exerted, but any influence is definitely not direct or totally predictable.

3. Some objects are more remote than others.

Related Concepts [External \*497]:

• Internal \*496

Type [External \*497]: Adjective.

## External Stakeholder: Concept \*495 October 27 2001

External stakeholders are stakeholders which are directly impacted by, or which use, a defined focus system.

For example: a product customer or user is an external stakeholder in relation to that product (a defined focus system).

Related Concepts:

• Focus system \*492

• Stakeholder \*233

• Internal Stakeholders \*494

Type: relation concept.

## Facilitator "facilitator" : Concept \*236. July 27, 2001

A person or group who assists others in carrying out a work process, such as quality control or setting objectives; by virtue of their being especially trained, qualified, and knowledgeable in that process.

## Fail Concept \*098 November 24th, 2004, 2019 .![]

‘Failure’ signals an undesirable and unacceptable system state.

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

A Fail limit specifies a point at which a system or attribute failure range begins. A single specified number (like Fail: 90%) is assumed to be a fail limit, the leading edge of a Failure Range.

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 \*098]:

• Fail Level \*098 (any point in the fail range)

• Fail Limit \*098 (the outer boundaries), not an exact synonym, but part of the set of fail levels.

• Failure Level \*098

• Failure \*098

• Warning \*098

• Must (Avoid) \*098: Historical usage only.

• Pain \*098 <- Jørgen Behrens, Symbian April 04

Related Concepts [Fail \*098]:

• Survival \*440

• Catastrophe \*602 ‘.’

• Range \*552: See ‘Failure Range’

• Must Do \*539: Historical usage only

• Limit \*606

• Level \*337

* OK \*669 added July 30 2014 tsg
* Tolerable \*539 added 080619

Keyed Icon [**Fail** \*098]: **!**

“In context on scalar arrows: ---!--->O---!--->

A Failure *Range* would use multiple Fail icons: ----!!!!!!--->->

A failure range, just above a catastrophe range, is icon-ed like *this:*

**O…………..…!!!!!!!!!!!!!!!!![-------------]>-------->**

.Catastrophe. !Failure! [Tolerable] >Success <- Ranges

The Success range beginning is denoted here with a single ‘>’, but we can also express the ranges using a series of keyed icons

**O…………..…!!!!!!!!!!!!!!!!![-------------]>>>>>>>>>**

.Catastrophe. !Failure! [Tolerable] >Success

Type [Fail \*098]: Constraint, Parameter.

## Failure Range. Concept \*546 May 29, 2003

A failure range is a scalar range where some stakeholders will not tolerate the attribute levels. They find the levels unacceptable.

A failure range lies between Acceptable Ranges and Catastrophe Ranges.

Related Concepts:

• Catastrophe Range \*603

• Acceptable Range \*545

• Success Range \*548

• Range \*552

Notes:

1. The range is primarily defined by a stakeholder, but can also be a range described by target and constraint requirements for a system, which are relating to the stakeholders’ ranges, perhaps as compromise requirements for a set of stakeholder ranges. (Insight due to Kai Gilb May 2002)

Keyed Icon [Failure Range, \*546] !!!!!!! 2 or more ‘!’.

Synonym[Failure Range, \*546]:

`• Fail Range \*546

• Intolerable Range \*546

Type [Failure Range, \*546]: Scalar concept

## False \*626 February 16, 2003

Predefined State. Same as Off.

Related To:

• State \*174

## Fault Concept \*339 June 5, 2003

A fault is a defect in a real system, which if ‘provoked’ by an agent or specific data, can result in a system malfunction.

Notes:

1. ‘Bug’ is a common synonym for ‘Fault’.

Example:

A geological fault is a defect in the earth. If the fault is provoked, the result can be an earthquake (a malfunction).

Figure \*339: From error to malfunction [Editor Note to Publisher: See Figure \*274, which is the same.]

2. Figure \*339 tracks the chain of events leading to a malfunction: a human engineer makes an error. This may be detected as a specification issue, which in turn, may be classified as a specification defect (which then may, or may not, be fixed). Alternatively, maybe the problem is not detected as an issue, and the error simply becomes a specification defect. A specification defect is in the system specification used to implement the system, it may become an actual fault in the implemented system. If the fault is then hit (provoked), a malfunction of some kind may well result.

Synonyms [Fault \*339]:

• Bug \*339

• System Defect \*339

Related Concepts [Fault \*339]:

• Error \*274

• Specification Issue \*529

• Specification Defect \*043

• Malfunction \*275

Type [Fault \*339]: Problem, Feedback.

## Feature: Concept \*519 January 23 2002

A feature is a class of stakeholder perceptible characteristics of a system, which are useful for comparing against competitive product, service or system benchmarks.

The point being to help stakeholders see that this system is equal, better or different from others. A feature can be performance attributes (easy to learn), cost attributes ( cheap to acquire) , functions (can send SMS) , or designs (uses titanium cover) . Features for one set of stakeholder (like repair services) will be different for another set of stakeholders ((like consumers)

Usage: in presenting proposed products or services or real ones. It has a positive connotation.

Related Concepts [Feature \*519]

• function \*069 not all functions are worth listing as features.

• design \*047 not all designs would be intelligible to the stakeholders as features ( too technical)

• performance \*434 or quality \*125 attribute, not all performance or quality attributes would be features. It depends on their levels compared to benchmark products or systems.

• cost attribute \*187 not all cost attributes would be features. It depends on their levels compared to benchmark products or systems.

## Fix Probability Concept \*067 November 21, 2002

The Fix Probability is the probability that an attempt to correct (fix) a specification defect, or system fault, will succeed.

Notes:

1. This probability should be based on past data concerning ability to remove defects.For example, IBM reported a 5/6 ‘fix probability’ for defects found by Inspection (SQC) in software engineering documents <- [M. Fagan, IEEE Transactions on Software Engineering. Vol. SE-12, No. 7, July 1986].

2. Higher than 82% fix probability has been recorded at NASA and IBM [(IBMSJ-1-94, Kan]. Under the condition that rigorous quality control of the each fix is carried out. The level is closer to ~99.9%.

## Focus System \*492 August 17, 2003

A focus system is the system, or product, we have decided to focus our attention on. All other systems related to it are ‘Environment’ systems.

The purpose of this concept is to allow us to define a focus on one particular element of a set of related systems.

For example if we are doing a project to develop a product, then that product is likely to be a focus system. All other related engineering and management processes and tools needed to do the project are the environment systems for the focus system.

Synonyms [Focus System \*492]:

System of Interest <- Mil-Std-499B

Related Concepts [Focus System \*492]:

• Environment System (Enabling Systems <-MIL-STD-499B) \*493

Type: ???

## Form Concept \*068 February 26, 2003

A form is a written document (electronic or paper) that guides people to give specific data with specific content in specific formats, in specific presentation sequences. It is a ‘work process standard’ for data collection. It is also an implicit ‘procedure’ for data collection.

Notes:

1. A form is something intended for direct data collection. It is usually fixed in format and appearance.

2. If an electronic form is enhanced with user dialogue; and has varied choices and computer presentation, then it becomes less of a ‘form’ and more of a human-computer dialogue.

Related Concepts [Form \*068]:

• Template \*254

• Standard \*138

• Procedure \*115

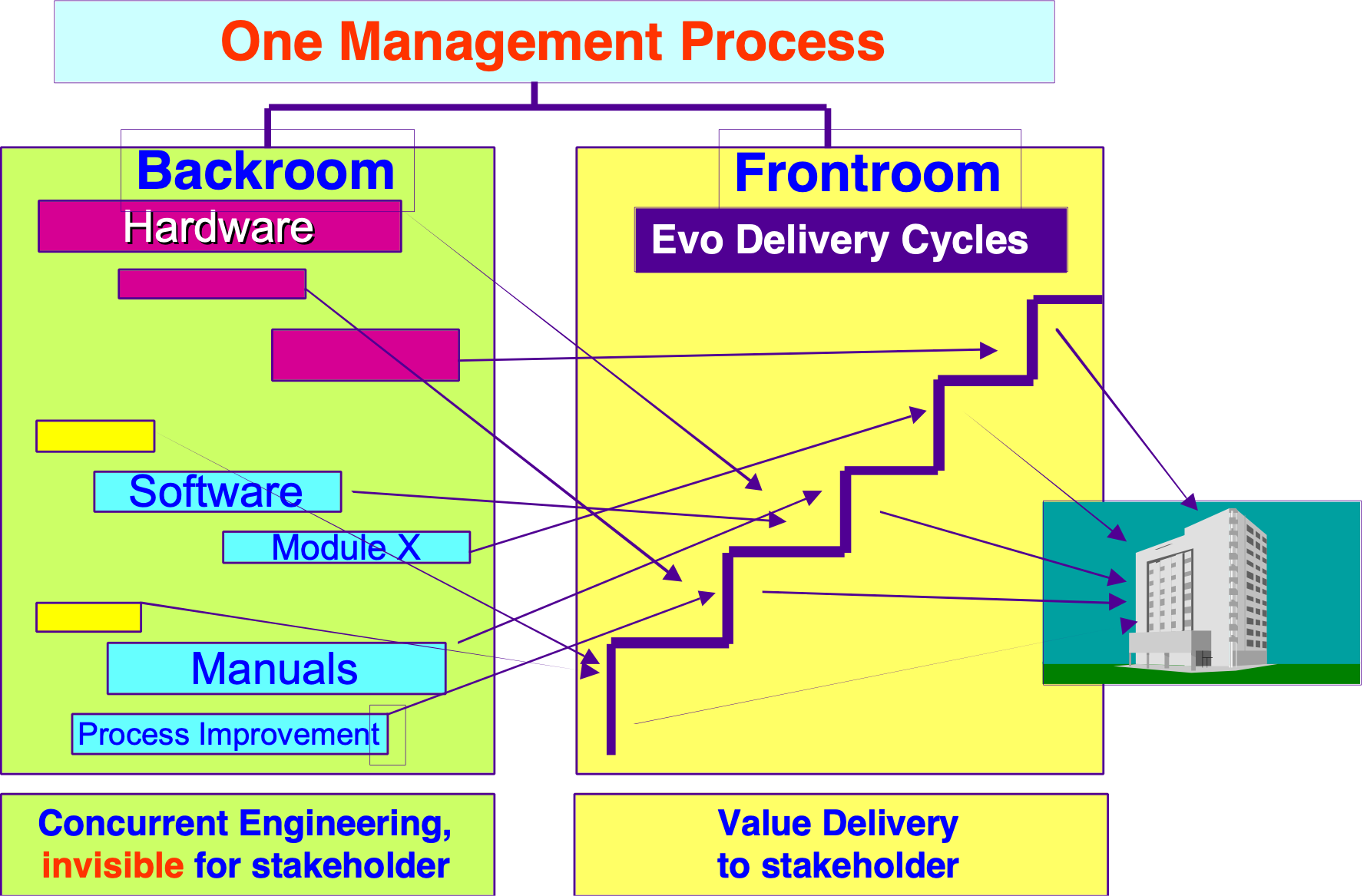
Type [Form \*068]: Tool [Data Collection].

## Frontroom Concept \*343 January 2, 2003

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.

Usage: Typically, used to refer to the delivery cycle part of the result cycle. The frontroom is where the step is delivered to the stakeholders.

The frontroom is where stakeholder-level results of the step integration can be tested and measured.

*Illustration compliments of Simon Porro, Eindhoven, SPI Partners, 2001. (Permission Granted)*

Related Concepts [Frontroom \*343]:

• Backroom \*342.

• Recipients (\*341): the targets of frontroom implementation activity.

Type: Adjective, Noun.

## Function (adjective) \*556 June 22, 2002

A function adjective is used to restrict the modified noun (constraint as in function constraint) to the function area.

## Function, To Concept \*557 March 18, 2003 B

‘To function’ means to carry out the defined functions of a system under the defined conditions, at least at survival level or better, for performance and cost.

Type [Function, To \*557]: Verb.

## Function Concept \*069 April 19, 2003 B

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.

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 sub-functions. 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 [Function \*069]:

• Attribute \*003

• Design Idea \*047

• Mission \*097

• Function Design \*521

Drawn Icon [Function \*069]: An oval (A circle would also be considered a function.)



Keyed Icon [Function \*069]: O or parenthesis, ( )In context: ----->(<Function Tag>)---->

Example: --------->O------> (a function with attached performance and cost attribute scales).

This describes a system, the function keyed icon, ‘O’, is combined with two scalar arrows.

Type [Function \*069]: Attribute, Parameter.

## Function Baseline. Concept \*489. May 6, 2004

A function baseline is an analytical statement of system functionality.

Usage:

It can be used to state the results of systems analysis of existing systems.

It can be integrated with future comparable functional requirements, so that it is easier to see the proposed functional changes. It is analogous to the scalar baselines.

Related Concepts [\*489]:

**• Benchmark \*007**

**• Baseline \*351**

**• Scalar Baseline \*490**

**Example: a pure baseline without a requirement**

Function X:

Type: Function Baseline.

PG: Baseline [2001, Product Y] {Game 1, Game 2}.

**Example: a baseline mixed with a requirement**

Function X:

Type: Function Requirement.

Baseline [2001, Product Y] {Game 1, Game 2}.

Description [201x, Product Y] {Game 1b, Game 3}.

Type [\*489]: specification.

## Function Constraint Concept \*469 March 30, 2003B

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).

Examples:

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.

Synonyms [Function Constraint \*469]:

• Function Restraint \*469

Related Concepts [Function Constraint \*469]:

• Function \*069

• Function Target \*420

• Requirement \*026

Keyed Icon [Function Constraint \*469]: [O]

or [ (Function Tag)] or (underlined) [ (Function Tag)]

Type [Function Constraint \*469]: Constraint.

## Function Design Concept \*521 April 23, 2003

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.

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).

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 Design \*521]:

• Function \*069

• Function Requirement \*074

• Design Process \*046

• Design Specification \*586

Type [Function Design \*521]: Design [Function].

## Function Requirement Concept \*074 May 11, 2004

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 at specified performance levels and costs 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.

6. 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.

7. NOTE: Functionality is one of the six characteristics for internal and external quality in ISO/IEC 9126-1 [ISO/IEC 25010]. Functionality requirements should not be confused with functional requirements. Functionality requirements specify the quality of the functional requirements and include requirements for the software product to be suitable, accurate, interoperable, secure and compliant with relevant functional standards and regulations. <- ISO/IEC JTC1/SC7 N2981 2004-01-22 Document Type 2 nd CD ballot Title Second CD Ballot, CD 25030.2: Software engineering – Software quality requirements and evaluation (SQuaRE) – Quality requirements

Related Concepts [Function Requirement \*074]:

• Function Target \*420

• Function Constraint \*469

• Function Design \*521

• Function \*069

Drawn Icon [Function Requirement \*074]: Any oval shape.

Keyed Icon [Function Requirement \*074]: (*Function Tag*) “The parentheses symbolizes the oval icon.”

An alternative is O.@ “A Function Target.”

Examples:

Example 1: (Voice Recognition).

Example 2: (Voice Recognition) -> Speech Input Quality.

*Speech Input Quality is a performance attribute impacted by the function requirement Voice Recognition.*

Type [Function Requirement \*074]: Requirement.

## Function Target Concept \*420 May 6, 2004

A function target is a specific qualified function requirement type.

A function target is specific testable function, it is not generic, nor is it a function constraint.

Example:

Propulsion Capability:

Type: Function Target. “Could also be called a Function Requirement.”

Description [Europe, Release 8.0]: 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].

Test: {Rough Terrain, Novice Driver}.

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 Target \*420]:

• Function Constraint \*469

• Function Requirement \*074

• Target \*048

• Function \*069

Keyed Icon [Function Target \*420]: O.@

Type [Function Target \*420]: Parameter, Target.

## Fundamental Constraints: Concept \*426 July 28&29, 2001

A ‘given’.

Fundamental Constraints are constraints that are handed down to us from higher authority. They are fundamental if we choose not to question them, but to heed them without question or alteration.

Note 1.: fundamental constraints are of various, and potentially conflicting, authority. The conflicting authorities can be ignorant of one another. There can be various degrees of power behind the various authorities.

Note 2. What are ‘non-fundamental’ constraints?

• anything which is specified but ignored with reference to higher priorities.

For example: a Corporate quality constraint (like minimum 99.9% availability) which is ignored because the current customer contract has a Limit of 99.98% availability.

• anything which is specified as a constraint by one stakeholder, but is, or might be, explicitly overridden by another higher priority constraint or requirement.

For example: a constraint on use of methods or processes which could be overturned by national law or international considerations. Maybe only if you ventured into these other market areas.

Keyed Icon [Fundamental Constraint] ----[---------]------ the square brackets.

Identical to the keyed icon for ‘Limit’ (a clear Fundamental Constraint).

Compare to -----[----------]----- Generic constraints icons (\*245).

Example:

R---------FC [------------------------->PL-------->O

R= a resource for example Engineering Hours. FC = a Fundamental Constraint, for example ‘maximum Legal Overtime Hours per Engineer per month’. PL = a Planned Level of Engineering Hours.

*Related concepts:*

*• Strategic Constraints, support Fundamental constraints. They are set at ‘our’ level of responsibility.*

*• Means Constraints. They are set by instances who are supporting ‘our’ level of strategic constraints.*

*Note: The {Fundamental, Strategic, Means} set is identical to the use of those terms for requirements and objectives in general.*

*• Limit parameter constraint is typically more fundamental than even a ‘Fail’ level objective.*

*• constraints are binary constraint borders. A constraint may, or may not, also be ‘fundamental’.*

*• Global Constraints: constraints which are system-wide, as opposed to only applying to specified sub-components. These can be ‘fundamental’ or not.*

*• Fundamental Objectives: ‘givens’ as objectives.*

## Fundamental Objective "fundamental objective" . Concept \*229 November 23, 2001 (+Churchill)

Fundamental objectives are ‘givens’. They are what we are trying to support and impact by reaching ‘strategic objectives’. They form the top level of a relative hierarchy of objectives. They are the given objectives handed down to ‘us’ from above.

Note 1. All lower-than-fundamental level objectives are part of ‘the design’ to reach these fundamental objectives.

Note 2. The ‘us’ (in above definition) is a defined level of the development hierarchy which receives fundamental objectives (from our ‘masters’). It is a level which defines and works towards our strategic objectives, in order to satisfy those fundamental objectives.

Related Concepts:

• Means Objectives: We may define some objectives that we believe will help us reach our strategic objectives; these are called ‘means objectives’.

• Strategic Objectives: objectives which are strategies for reaching the fundamental objectives.

Rationale 1: The concept of fundamental objectives is useful because it helps us organize various priorities of objectives. The fundamentals are the ones we cannot question; just serve.

Synonym: (almost) Fundamental Requirement. Not fully, since objectives do not contain the concept of constraints, and requirements do. So a Fundamental Constraint is a Requirement but not an objective.

Keyed Icon [Fundamental Objective \*229] any requirements or objectives icons can be used to represent a fundamental objective. If you want a direct iconic representation of the ‘Fundamental’ then I suggest [@] a totally constrained ‘[…]’ goal ‘@’.

*Quotation*

“The fundamental objectives hierarchy specifies in detail the reasons for being interested in a given problem. For each of the fundamental objectives, the answer to the question ‘Why is it important?’ is simply ‘It is important.’ ”

Source: [KEENEY92 page 78]

Type: Requirements.fundamental, a hierarchy class

The term ‘fundamental objective’ was coined by Keeney [Keeney92].

“I would say to the House, as I said to those who have joined this government, that I have nothing to offer but blood, toil, tears and sweat. We have before us an ordeal of the most grievous kind…. You ask, what is our policy. I will say it is to wage war, by sea, land and air, with all our might and with all the strength that God can give us: to wage war against a monstrous tyranny, never surpassed in the dark, lamentable catalogue of human crime. That is our policy. You ask, what is our aim? I can answer in one word: It is victory, victory at all costs, victory in spite of all terror, victory, however long and hard the road may be; for without victory, there is no survival.”

Winston Churchill, War Papers II, p. 22. From his address to the Commons 1940 after forming his first government. In Jenkins, Churchill, p.591.

The fundamental objective is survival. Victory is the strategic objective. War is the means objective. “blood, toil, tears and sweat.” Is one resource.



## Funder: Concept \*344.Funder. July 28, 2001

One who provides money, or other resources, for a project.

## Fuzzy Concept \*080 January 9, 2003

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. Example above.

Keyed Icon [Fuzzy \*080]: < *fuzzy term* >

4. A fuzzy specification essentially amounts to a declaration by the writer that the specification is defective at that point.

Type: Specification Concept, Keyed Icon.

## Gaming "gaming" : Concept \*241. July 28, 2001

Conscious manipulation of requirements or evaluations so as to achieve special aims.

Source: Keeney92, page 96.

## Gap Concept \*359 March 5, 2003

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).

Note: In general the larger the gap, the greater the need to deal with it (‘higher 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’.

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.

Synonym [Gap \*359]:

• Residue \*359

Related Concepts [Gap \*359]:

• Range \*552

• Design Problem \*654 : A gap defines a new design problem.

Keyed Icon [Gap \*359]: === “Any length of two or more ‘=’.”

In context, along a scalar arrow ---<••••|====>---->O---<••••••|=======>------>

where ‘|’ is the current benchmark level and ‘•••’ represents a range already achieved.

Type [Gap \*359]: Scalar Design Data.

## Generate . Concept \*392 Spawn/Create/Generate/Produce. Aug 8th, 2001

‘Generate’ is a relationship concept. It relates things that produce other things.

Keyed Icon [Generates, \*392] {

Example: ‘A { B’ means ‘A Generates B’.

Note [Keyed Icon Example]. There is an implication that the Generator (A) continues to exist after it has generated ({) something (B). This is distinctly different from ‘A=> B’ (or A Before B) which states that A proceeds B. But it does not mention responsibility for creation, nor is there any implication that A exists after ‘=>’.

Note Antonym ( ‘}’, \*393) : A}B = ‘A is condensed into B’ ..

Usage [Generates Icon] the format a {b, c} can be used to indicate a set of one or more components {b and c} which are spawned from ‘a’. Example:

Mom {Alex, Andy, Andrea}. The use of the ‘set’ parenthesis is entirely natural.

Synonyms: Spawns, Creates, Explodes Into, Produces.

Antonym: \*393 Compress/Condense/Reduce To/ Simplify/Combined/Summarize

*Examples:*

Mother { ChildrenMother Generates ChildrenInventor { {Product, Patent} Note: not = to ---> Inventor {Product, Patent} Process Owner { Process ImprovementTeam {John, Abdul, Aaron, Solveig} { Product {Software, Hardware, Training}.Goal [IF Plant { Toxic Waste] 6,000.

Example note:

• Inventor { {Product, Patent} means Inventor actively generates {Product, Patent}

• Inventor {Product, Patent} means that Inventor is the Name Tag of the set of things known as {Product, Patent}

Related Concepts:

\*393 Compress/Condense/Reduce To/ Simplify/Combined/Summarize

## Generic Concept \*081 January 11, 2003

Generic means *applicable to any individual component* of a group of components within a defined scope.

Generic means general, as opposed to specific. ‘Generic’ things are often ‘reusable’ (with local tailoring that makes them more specific) in a variety of situations.

Notes:

1. We use the adjective ‘generic’ when we refer to any systems engineering specifications (for example, rules, processes, conditions and, requirements specifications), which can be used, perhaps with tailoring, in many different kinds of specification processes and/or specifications. For example: generic rules, generic exit conditions, generic development process, and generic constraints.

2. The main idea behind ‘generic’ things is reuse. We want to reuse methods, rules, and any form of engineering and management wisdom. The alternative is to have repetition of methods and ideas, which are in fact common in many areas. This would lead to problems in consistency, learning and updating. It would also increase unnecessary bureaucracy: instead of having to write, learn and maintain multiple versions of essentially the same thing, we do it once.

Related Concepts [Generic \*081]:

• Specific \*136

Type: Adjective, Scope.

## Generic Attribute Level Icon: (Concept \*251) July 29, 2001

A neutral graphic symbol, (---|---) for a numeric level of a performance or cost attribute.

Usage: The generic symbol is used in place of unknown, undetermined or uninteresting specific symbols of attribute priority or character; such as any benchmark (example ---<<--) or target (example ---->---).

Rationale: to avoid being more specific about the benchmark or target type.

Type: generic icon specification convention. {Planguage.Attribute Level Icon Point.Keyed Icon}

Related Concept: \*252 Attribute Level Icon Point.

Usage 2. The *vertical line* (‘|’) can be used in both keyed and icons. It can be used at one or more points on the attribute arrow lines (---->O---->) to indicate levels of the attribute.

(O----|-----|-->).

It can be used to indicate ‘no specific type’ of benchmark or target.

Or it can be used with a supplementary text indicating type of attribute level, such as ‘Goal’ or ‘Wish’ level. Example:

O----|Goal [SF, Next Year] 30%-->

## Generic Constraint: (Concept \*245) July 29, 2001

A generic constraint applies to a defined scope. It is tailored in interpretation or specification at the detailed level.

For example:

•(1) a corporate policy constraint which applies to all corporate projects:

Corporate Quality 2. All quality levels shall be clearly better than our competitors.

Note that the precise quality levels needed will depend on the competitor levels, which will be unknown initially and will vary in time. So this constraint requires tailoring; and is dynamically changing through system lifetime.

•(2) a contractual requirement which applies to the entire contracted project.

Contract 2.3.4: all system components must be legal import in all European Markets.

•(3) a legal constraint which applies to products made in your country.

Euro Law 2001-3.4-334: No components, including imported components can use illegal Child Labor, even if assembled in another country by a supplier or customer.

Type: Requirements class, parameter, {requirement.constraint.generic}

Synonyms:

• ‘system level requirement’,

• constraint (often used as a short form of Generic Constraint; while specific (\*136) constraints are then referred to as ‘requirements’).

• Derivative Source (\*450): a Generic Constraint’s generic name!

Keyed icon [Generic Constraint, \*245] : ---X {----}Y---

(a conscious reuse of ‘set’ parenthesis)

The set parenthesis (‘{…}’) can be used singly, or together, when placed on an attribute arrow icon scale (--{----->).

Note 1. Icons: there is no conflict with use of { } for other reasons ( set , spawns, combines) because the Generic Constraint icon is only used on an attribute scale (---->).

Note 2. Icons: compare the use of ----[----]---- as keyed icons for Limit, which is a Fundamental Constraint.

Note 3 Icons: The simultaneous existence, as a requirement specification, of both Fundamental Constraints and Generic Constraints is quite possible. One Constraint specification can even be both (Fundamental and Generic). They might even apply simultaneously (Time qualifier)and with same qualifiers otherwise (Place, Event)

----------F1[-------GC1{---------GC2{-----------}GC3-----------}GC4-----------]F2------->

In specific instances, the icons of specific synonyms such as Fail (->>) and Goal (->) can be applied to express Generic Constraints..

The open side’ (this side}’ or ‘ {this side)’, is the direction of acceptable attribute levels. ----NO {-----OK-----}NO-------

Rationale [Generic Constraint]: this classification is used to emphasize the fact that the actual practical constraint applied at the local level of specification will need to be tailored to that specific level. It will have to be ‘generated’ depending on the Generic Constraint specification, and the corresponding local context. This is the ‘derived constraint’ (a specific local constraint derived from a generic constraint).

Notes:

Note 1. Consequence Analysis. Generic constraints are generally (not always) set by powerful stakeholders and are generally not subject to much negotiation. Of course they can be in conflict with each other, and with any other requirements specifications or system environment realities. So they need to be examined in terms of their design consequences, costs and qualities. If necessary, some feedback, to the stakeholders concerned, will be necessary, and may result in modification, ignoring or removal of the constraint.

Note 2. Outer Border. A generic constraint is a specification, which ‘draws a border’ regarding requirements or design, but which allows more-specific requirements (which can be specific constraints on requirements or design) to be specified inside that border.

Note 3. Probably Global. Generic constraints usually consider interests, which are broader or more fundamental than a specific project or product (or other ‘lower level’ or ‘shorter-term’ perspectives). For example interests of a product line, or national laws and customs. To the degree they have wide application, they are also classified as Global.

Note 4. Conflict Analysis. In cases of a conflict between any generic constraint and any other specification, a generic constraint probably has a priority which is higher than any conflicting specific requirement or design. But this entirely depends on the exact nature of the information about the requirements which determine priority {Authority, Source, Priority, qualifiers [such as when, where and event conditions] }. To the degree which it does have priority over others, it is also classed as a Fundamental Constraint.

Related Concepts [Generic Constraint]: ‘applies to defined scope, and will be tailored locally’

• Constraint , \*218.

• Requirement,

• Fundamental Constraints. Of basic unquestionable authority and priority.

• specific constraint.,

• Global Constraint. \*245: of wide scope, entire defined system scope, local tailoring not generally necessary.

• Generic (\*081), defined scope, reusable, tailored, local interpretation.

Sub-categories [Generic Constraint]:The potential Generic Constraint sub-categories are entirely open to the needs of the project. But there are some categories which are typical and frequent. These are the same as for any constraint, including specific constraints and global constraints.

Examples:

Function, Performance and Cost: which mirror the three main other categories of requirements. The distinction here is that the Generic Constraint specifications are defined a level outside of the specific areas that they impact.

Other categories: Legal, National, Market area, Competitive, Partnerships, Product Line Architecture, Design (\*181) , Interfaces, Cultural, Ethical, System Component Constraints, Financial, Large Customer constraints, System-wide, Corporate Policy, Standards, and as many more possibilities, as you find useful.

Example:

Corporate Reliability:Type: Generic Quality Constraint. Scale: Mean Time Between Failure (MTBF). Limit [All Corporate Branded Products] 30,000 hours <- Corporate Quality Policy.

This would make invalid any attempt to specify, as a specific quality requirement (example below):

Product Reliability: Type: Quality Requirement.Scale: MTBF.Goal [Product XX] 20,000 hours <- Customer Contract.Wish [Product Next Generation] 40,000 hours <- Marketing Long Term Plan.

The above requirement would have to be adjusted to the minimum of 30,000 hours MTBF in order to satisfy the corporate minimum quality

(the Generic Quality Constraint, which due to the qualifier, and the ‘Limit’ parameter, has higher priority, and could also be classed as a Fundamental Constraint)

for Product XX and not hurt their brand image

But, the specific requirement for ‘Product Next Generation’ would not need to be adjusted to comply with the current corporate generic requirement, as it is an improvement (40k>30k) over the corporate criteria.

We could explain this adjustment to people who might not otherwise understand why we had increased the specification beyond the customer’s wish.

Example:

Product Reliability:Type: Quality Requirement.Scale: MTBF.Goal [Product XX] 30,000 hours <- Customer Contract & Corporate Reliability.Constraint: Corporate Reliability.Stakeholder: Corporate Quality Director.

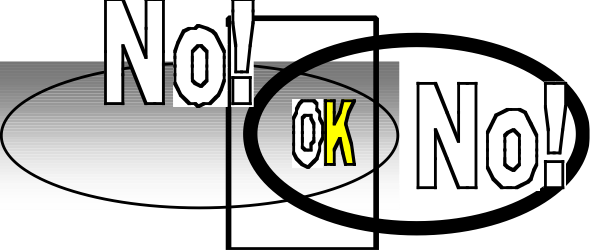


Figure: three simultaneously applicable generic constraints, leave one area of common “OK” territory for more specific decisions to be made, such as specific requirements and designs.

Some other examples.

GF1: Type: Generic Function Constraint: Defined: The product must be a mobile telephone; and not wander into replacing or supplementing computers or entertainment or automated guidance systems.

Rationale: this would impact other of our business areas and partnership agreements with Company X and Company Y. <- Corporate Product Policy Feb 20 2005. Paragraph 33.4

GF2: Type: Generic Function Constraint:

Defined: The Mission is strictly limited to scientific exploration of our own planet, and must not deal with other planets or solar systems.

Rationale: this is NASA and JPL territory.

GSC1: Type: Generic System Constraint:

Defined: No aspect of the system can threaten sister company products, or national laws in any way: this includes quality, costs, targeted markets, technology selection and manufacturing and disposal processes; but is not limited to the foregoing list of potential specific concerns.

Rationale: Company Product Development Policy <-- CDP Feb 20 20051, 3.44.

Design Constraints

(Architecture)

Function: ‘Our Product’’

Quality/Value

Requirements

Cost/Resource

Requirements

Architect

Design

Engineer

Product

Manager

Illustration text: Architects make system-wide decisions, which must be respected by all other engineers. For example, ‘system interfaces’ to enable safe, fast {change, tailoring, extension, development steps}. These are generally both generic constraints, and fundamental constraints. Design engineers try to reach quality targets, within the specified cost requirements, for specific subsystems.

Case:

Design constraints (\*181) may also be a type of generic constraint.

For example, on Howard Hughes’ ‘Spruce Goose’ aircraft (which was on public display in Long Beach, California) he was given a wartime design constraint, due to wartime metal shortages, of minimizing use of metal in the plane’s design\*\*.

So he mainly used wood (‘spruce’). He respected his design constraints. The wingspan was about 2.5 times that of a jumbo jet. Yes, it flew; once!

\*\* My father, Tyrell T. Gilb (1916-2001) had the job of the detailed engineering of the plane for Hughes to reduce metal content, he told me. <- TG.

*Example:*

GC: Type: Generic Quality Constraint. All <product> qualities must be <obviously better> than all our <competitors>.

This is a constraint on how low any specific quality level can be planned.

Notice this level varies with the competitors’ qualities.This statement might also be directly specified as a Fail or a Limit level level specification for all quality requirements. But, more likely a set of different Fail requirements will be set within the framework provided by the generic constraint ‘GC’.

## Generic Objectives "generic objectives" : Concept \*216. July 29, 2001

A template objective, that is basic to many situations, and can be used as a checklist item to remember to include this objective. It is also a basis for tailoring that objective more precisely to the current situation.

“Whereas strategic objectives are intended to define the concerns of a single decisionmaker for all decision situations, generic objectives attempt to define the concerns of all decisionmakers in a single decision situation.”

Source: Keeney, page 63.

## Generic Qualifier, \*668 July 21 2014

A Generic Qualifier, is a set of higher level concepts, which are going to be decomposed into specific subsets and combinations, for use as qualifiers for any Planguage statement.

Scale Qualifiers are a particular type (for Scale) of Generic Qualifiers.

Here is a way of expressing the planned decompositions:

**Strategy A**. [Country, City, Target, Product Line, Service Level]

**A1** [Country = UK, City = London, Product Line = Magic, Service Level = None ] 50% of Planned revenue

**A2** [Country = USA, City = Los Angeles, Product Line = Magic Version 2, Service Level = 24 Hour Help ] 30% of Planned revenue

**A3** [Country = Norway, City = Oslo, Product Line = Magic Version 2, Service Level = {24 Hour Help, Norwegian Language}] 15% of Planned revenue.

The first line, Strategy A, lists the aspects of decomposition (a **Generic Qualifier**]. The following 3 examples of decomposition, select the decomposed sub-strategies, and document them. Together with an estimate of 1 dimension of effect (Planned revenue) each is expected to contribute.

## Generic Requirement "generic objectives" : Concept \*216 July 29, 2001

A generic requirement is one intended for potential use in many systems or sub-systems. It is always reusable, and often tailorable.

Note 1. A template of the requirement is used as the basic start point for tailoring any instance of that requirement more precisely to a specific situation.

Note 2. Checklists of generic requirements are useful to ensure that they are used wherever relevant.

Note 3. It can be of several types including constraint and template.

Note 4. One powerful tactic for specification of a generic requirement is to use Scale Qualifiers.

Example [2 scale qualifiers]:

Scale: hours for [User Type] to successfully complete [Task].

The term ‘generic objective’ was coined by Keeney [KEENEY92].

“Whereas strategic objectives are intended to define the concerns of a single decision maker for all decision situations, generic objectives attempt to define the concerns of all decision makers in a single decision situation.”

Source: [KEENEY92 page 63]

Synonym: Generic Objective, template requirement.

Keyed Icon: <@> (Fuzzy Requirement).

Type: requirements hierarchy class. {Requirement.generic}

## Generic Standard (adjective) "generic Standard (adjective)" : Concept \*150. July 29&30, 2001

A 'generic' standard is an example, model, specification, form, or template of a work process standard, which can then be tailored, or applied without tailoring, by their user to suit their purposes.

Synonym: Work Process Standard, \*138 (sometimes equivalent, see below).

*Note 1. There are generic standards for forms, for meters, policies and other useful artifacts in this book.*

*Note 2. Generic standards are intended to be used (read, applied, taught from) directly by an organization.*

*Related Concepts:*

*• Template, \*254: An example or ‘model’ of something, which can be used to help people to tailor, or make something, based on that model. A template can be for something other than a ‘standard’. A generic standard can be presented in a form other than a template (for example a form or list or model). Generic templates are intended to be ‘filled out’ or ‘copied’, or more or less copied, with small modifications locally, and used by a local user. They are but one type of generic standard.*

*• Work Process Standard, \*138: A work process standard is an ‘official’ written specification that guides a defined group of engineers in doing a process. It is a best-known practice.*

## Generic Term: Concept \*377. July 30, 2001

A generic term is a Term in a Specification that needs further definition, there or elsewhere.

*Related Concepts:*

*• Specific (\*136): ‘Specific’ means applicable to a limited, specified set of things.*

*• Generic (\*081): Generic means applicable to a large group or set of things. It implies that the things are ‘reusable’ in a variety of situations. Needs further definition.*

*• Term (\*151): A set of words and/or symbols (including icons) with a Planguage-defined meaning.*

*For example (1), Scale Qualifiers (\*381) are one example of generic terms: this could be a Generic Standard (\*150) – a template for a ‘User Reaction’ definition.*

User Reaction: Scale: Probability that defined [People] will report defined [Faults] within defined [Time Limits].

*Then somewhere, just here locally, or in a more global Project Glossary, we define the Generic Terms*

*For example (2): ( and these could also be either Generic Standards, or specific to the project. ‘….’ Indicates a ‘list of more detail’, which we choose not to show here.*

People: Defined: The class of customer, {Customers (Default), Trial Users, …}.Faults: Defined: The class of Faults, {Show Stoppers, Help Problems, … }.Time Limits: Defined: Time from Fault occurred until report received and logged by us. {n Hours, N Days, Same Day}.

*• In the example (1) ‘defined [People]’ and two other similar qualifiers in the Scale Specification, are Generic Terms. That means, they need further definition.*

*• They are defined in example (2), and a set of known values for each is listed in the {Set Parenthesis}. Even this definition (2) is a Generic Term, because it needs further definition to pick the exact class or time.*

*• One People class (2), (‘Customers (default)’ ) is identified as the ‘Default Value’ (\*447).*

*The use of these terms is illustrated by the example (3): using* Scale Variables (\*446)

User Reaction:Scale: Probability that defined [People, Default: Customers] will report defined [Faults] within defined [Time Limits].Fail [Customers, Show Stoppers, 24-Hours] 90%Goal [ People = Trial Users, Faults = Help, Time Limits = 1 hour] 99%Wish [ Faults = Show Stoppers, Same Day] 95%

*Note [Illustration 3].*

*• We have illustrated defining the Specific Terms by simply using defined Term Classes (‘{Show Stoppers, Help Problems}’), and even doing so (3, Fail) in the same sequence (‘People (Faults (Time Limits’)) in which the Generic Terms were specified, in the Scale specification (1 above).*

*• We also illustrated (3, Wish & Goal) the use of Explicit Parameters (example ‘Faults =’) for defining Specific Terms (example ‘Help’).*

Reliability:

Type: Quality Requirement.

Scale: Mean Time Between Failure from defined [Event Beginning] to defined [Event Ending] for defined [System] under defined [Circumstances].

Meter: defined [Test Circumstances {Development, Acceptance Test, Operation}] <specify or refer to test procedure or protocol or standard>

Goal [<when, where, which circumstances>] <a level such as 30,000 hours> ( <Source.>

Authority: <which levels of management, or customer contract, determine the level?>

This is a simple example of a generic template for a quality requirement. The terms in <fuzzy> are template ‘hints’. Instructions as to what to fill out. The intent is that they be removed from an electronic template when used. Both the Scale Qualifiers (like ‘Event Beginning’) and the hints (like <when>) are Generic Term examples.

## *Gist ∑ Concept \*157 January 3, 2003*

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

Rationale:

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).

Examples:

Gist: All the functions related to transportation of people.

Software Interfaces:

Type: Architecture.

∑: The total set of software interfaces needed for this project.

Notes:

1. The detailed specification may already exist, or it may be made on the basis of an agreement about the Gist.

2. When summarizing a *scalar* specification, use the more specific parameter ‘Ambition.’

Synonym [Gist \*157]: Summary \*157.

Related Concept [Gist \*157]: Ambition \*423.

Keyed Icon [Gist \*157]: ∑

“Greek Summa, mathematical summary symbol.”

Type: Parameter.

## Global Concept \*375 January 11, 2003

‘Global’ is an adjective used to specify that something applies to all (each individual component) of the set of components within the specified scope.

Notes:

1. A global specification applies across all the components within its declared scope.

For example: a Reliability specification declared as global for a system will have implications for all system components which contribute to the system reliability.

Related Concepts [Global \*375]:

• Local \*376

• Inherited \*516 “Global specifications lead to inherited specifications.”

Type: Adjective, Scope.

## *Global Constraint Concept \*245. July 30, 2001*

A Global Constraint applies for a defined system scope, unless overridden by higher priority specifications.

Type: requirements class, parameter. {Specification.Requirement.Constraint.Global}.

Notes:

Note 1:Even a global constraint is limited in effect by all its attached priority information {Qualifiers, Authority, Scope (\*419)}. For example, a global constraint may be relative to a defined set of stakeholders. It is not absolute for the universe of all stakeholders. This is part of the system scope in its definition. For example, the constraint may apply only to ‘User = School Teachers’.

Note 2: A global constraint only has a level of authority which is specified, or implied (by Sources for example) in its Scope. It must compete for priority with Authority, and other priority-determining specifications, throughout its scope. It does not have higher priority automatically, just because it is global in scope.

Note 3: a global constraint may apply to a wider systems area than other constraints and other requirements. But the areas it applies to are defined by qualifiers or other suitable parameters describing it. For example by Depends On, Impacts, Authority.

Synonym: system level requirement, constraint (often used as a short form of Global Constraint, local constraints are then often referred to as ‘specific requirements’).

Keyed Icon: ---X [----]Y--- . X and Y are tags of Global Constraints.

In specific instances the icons of specific synonyms such as Fail (->>) and Goal (->) can be applied.

The qualifier parenthesis (‘[…]’) can be used singly or together when placed on an attribute arrow (‘->, ‘----’)..

The open side ‘this side]’ or ‘ [this side’ is the direction of acceptable attribute levels. The

Notes: (continued)

4. Global Constraints are often set by powerful stakeholders and are then generally not subject to much negotiation. Of course they can be in conflict with each other, and with business needs, so they need to be examined in terms of their design consequences, costs and qualities. If necessary, some feedback, to the stakeholders concerned, will be necessary, and may result in modification, overriding or removal of the constraint.

5. A Global Constraint is a specification, which ‘draws a border’ regarding requirements or design, but which allows more specific requirements (which can be specific constraints on requirements or design) to be specified inside that border. For example: ---GC[----->>Fail-------->O

6. Global Constraints usually consider interests, which are broader or more fundamental than a more-local project or product (or other ‘lower level’ or ‘shorter-term’ perspectives). For example, interests of a product line, company policy or national laws and customs.

7. In cases of a conflict between any Global Constraint and any other specification, a Global Constraint usually has a priority, which is higher than any conflicting specific requirement or design. But the qualifiers and other parameters such as Authority will determine its real priority in particular instances.

Related Concepts [Global Constraint]: ‘defined system scope’.

• Constraint \*218, “All project components” for example.

• Generic Constraint, \*245: ‘A generic constraint applies, and must be tailored locally, to a defined class or group.’ . “All Quality specifications” for example.

• Requirement,

• Fundamental Constraints. Of basic unquestionable authority and priority.

•  Local Constraint.

•  Local (\*376) specification. ,

• Scope (\*419), Global Scope ( see \*419), Local Scope (see \*419)

Sub-categories [Global Constraint]:

The potential Global Constraint sub-categories are entirely open to the needs of the project. But there are some categories, which are typical and frequent.

Examples:

1. Performance/Quality and Cost constraints.

The distinction here is that the Global Constraint specifications have wider scope than any of the non-Global Constraint specifications.

2. Other Global Constraint categories: Legal, National, Market area, Competitive, Partnerships, Product Line Architecture, Design, Interfaces, Cultural, Ethical, System Component Constraints, Financial, Large Customer constraints, System-wide, Corporate Policy, Standards, and many more possibilities, as you find useful. Again the distinction is wide scope.

Example:

Corporate Reliability:

Type: Global Quality Constraint.

Scale: Mean Time Between Failure (MTBF).

Fail [All Corporate Branded Products] 30,000 hours <- Corporate Quality Policy.

This would make invalid any attempt to specify, as a local quality requirement:

Product Reliability:

Type: Quality Requirement.

Scale: MTBF.

Goal [Product XX] 20,000 hours <- Customer Contract.

Wish [Product Next Generation] 40,000 hours <- Marketing Long Term Plan.

The above requirement would have to be adjusted to the minimum of 30,000 hours MTBF in order to satisfy the corporate minimum quality (the Global Quality Constraint) for Product XX and not hurt their brand image. But, the local requirement for ‘Product Next Generation’ would not need to be adjusted to comply with the current corporate Global requirement, as it is an improvement over the corporate criteria.

We could explain this adjustment to people who might not otherwise understand why we had increased the specification beyond the customer’s wish.

Example:

*Product Reliability:*

*Type: Quality Requirement.*

*Scale: MTBF.*

*Goal [Product XX] 30,000 hours <- Customer Contract & Corporate Reliability.*

*Constraint: Corporate Reliability.*

*Stakeholder: Corporate Quality Director.*

OK

No!

No!

Figure: three simultaneously applicable Global Constraints, leave one area of common “OK” territory for more specific decisions to be made, such as specific requirements and designs.

Some other examples:

GF1: Type: Global Function Constraint:

Defined: The product must be a mobile telephone and not wander into replacing or supplementing computers or entertainment or automated guidance systems.

Rationale: this would impact other of our business areas and partnership agreements with Company X and Company Y. <- Corporate Product Policy Feb 20 20xx. Paragraph 33.4

GF2: Type: Global Function Constraint:

Defined: The Mission is strictly limited to scientific exploration of our own planet and must not deal with other planets or solar systems.

Rationale: this is NASA and JPL territory.

GSC1: Type: Global System Constraint:

Definition: No aspect of the system can threaten sister company products, or national laws in any way: this includes quality, costs, targeted markets, technology selection and manufacturing and disposal processes; but is not limited to the foregoing list of potential specific concerns.

Rationale: Company Product Development Policy <-- CDP Feb 20 20xx, 3.44.

Design Constraints

(Architecture)

Function: ‘Our Product’’

Quality/Value

Requirements

# Cost/Resource

# Requirements

Architect

Design

Engineer

Product

Manager

Illustration text: Architects make system-wide decisions, which must be respected by all other engineers. For example, ‘system interfaces’ to enable safe, fast {change, tailoring, extension, development steps}. Design engineers try to reach quality targets, within the specified cost requirements, for specific subsystems.

*Example:*

GC:

Type: Global Quality Constraint.

All <product> qualities must be <obviously better> than all our <competitors>.

This is a constraint on how low any specific quality level can be planned. Notice this level varies with the competitors’ qualities.

This statement might also be directly specified as a constraint specification for all quality requirements. But, more likely a set of different constraint requirements will be set within the framework provided by the Global Constraint ‘GC’.

## Goal: Mål [No]: Commitment: Concept \*109 June 4 2014, 19 Oct 2014

A goal is a primary numeric target level of performance.

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

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

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

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

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

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).

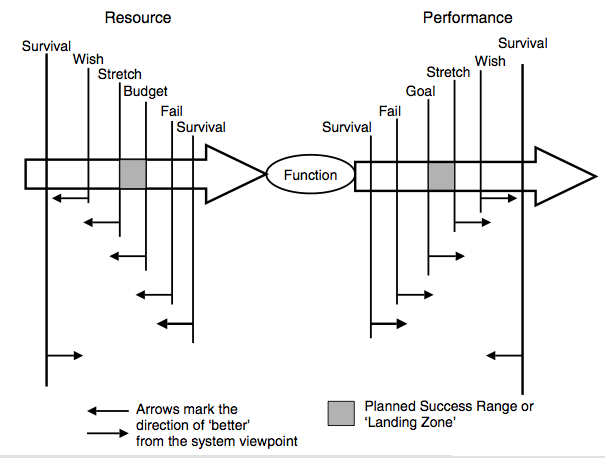
3. I now prefer the term ‘Goal’ instead of my traditional ‘Plan’ parameter. ‘Plan’ refers to so many other elements of planning. If an alternative were needed for Goal, I would use the more explicit ‘Planned Level.’

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!”



Here are some conditions for a Goal statement to be valid:

Technically possible - within state of art

Economically Possible - resources exist

Costs consistent with other Requirements

Effective, and effect necessary to satisfy stakeholder needs

Profitable: value over cost

Prioritized: by any rules of priority, such as

Effectiveness

Profitability

Politics

All [Conditions] in the Goal statement are ‘true’

**Conditions for A Goal Level. When is a goal level really valid?**

Synonyms [Goal \*109]:

• Plan \*109 (historic usage only), too general, since all targets and constraints are plans.

• Planned Level \*109 (historic usage only)

• Goal Level \*109 (see Level \*337)

• Planned Goal \*109

• Target [Intel, 2003], problem here is PL uses Target \*048 generically

• Mål [Norsk Bokmål] \*109

• Committed \*109 Suggested by Rob?, BCS Pmgt Class Oct 2014

• Promised \*109 added Oct 2014 Tsg

Related Concepts [Goal \*109]:

• Aim \*001

• Target \*048

• Stretch \*404

• Wish \*244

• Ideal \*328

• Objective \*100

• OK \*669

Keyed Icon [Goal \*109]: >

“A single arrowhead, on a performance arrow, pointing towards the future. 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.”

Type [Goal \*109]: Parameter [Scalar], Target.

## Goal Step: ----)--- Concept \*620 January 3, 2003

A goal step is an intermediate target level on the way to an agreed goal level.

Synonyms (\*620, Goal Step):

• Intermediate target

• Intermediate goal

Keyed Icon (\*620, Goal Step) ‘)‘ (in context -----)--->---->)

Noted this symbol is consistent with other target icons in pointing to right (future)

Domain: Requirements. Targets. Goal Steps

Related Concepts (\*620, Goal Step)

• Evo step \*141

• Incremental Impact \*220

• Now level \*513 ---(----

• Delivered Level \*533 ---•---•---

## Grammar "grammar" : Concept \*160. July 31, 2001

Implicit: Planguage grammar. The system of rules used by Planguage users to form statements and larger constructions or relations from Planguage elements. The rules for Planguage.

## Guideline Concept \*460 March 13, 2003 D

A guideline is specific written piece of advice, a recommendation, a tip, a hint, a suggestion, or a warning. It can be found in a specification, but is often placed into a set of guidelines, which are more widely available (sometimes held alongside the standards documentation).

Guidelines are not standards; they do not have absolute power. Their qualifiers and other parameters, such as ‘Authority’ and ‘Priority’ can give more information as to their individual priority.

Rationale: We need a term somewhat stronger than a comment, and somewhat weaker than a standard.

Example:

Goal [USA, Fall Season]: 60,000 Units.

GP: Guideline [USA]: We should normally plan to produce as much as we expect to sell.

Authority [Guideline]: General Sales Policy [North America].

Priority [GP]: This does not replace actual current production plans, which may consider stock on hand and business downturns, when setting production targets.

Synonyms [Guideline \*460]:

• Advice \*460

• Recommendation \*460

• Tip \*460

• Hint \*460

• Suggestion \*460

• Warning \*460

Related Concepts [Guideline \*460]:

• Standard \*138

• Note \*018

Type [Guideline \*460]: Guidance, Parameter.

## Handbook Variant: Concept \*201. July 31, 2001

A variation of the Planguage Handbook made for a particular organization, or language.

## High-Level Concept \*082 August 16 2008

‘High-level’ refers to a position that is ‘high’ (that is, closer to the top than the bottom) in any specified hierarchy of system components.

Since a given hierarchy can be made using any arbitrary top and bottom notions, the reference applies only to a given hierarchy, and is not universal for the system.

Notes:

1. I would prefer if the term, ‘high-level,’ were not used in specifications because it is too imprecise. Specific reference ought to be made to the component(s) being referenced. For example, rather than specify a ‘high-level authority,’ specify the ‘Divisional Director.’

2. ‘High-level’ does not automatically translate to high priority in system implementation terms.

Related Concepts [High-Level \*082]:

• Upstream \*291

• Downstream \*052

• Priority \*112

• Authority \*005

Type [High-Level \*082]: Adjective [Relationship].

## Hierarchical Concept \*083 February 1, 2003

Hierarchical is an adjective used to describe ‘belonging to a hierarchy’. The elements of a hierarchy can be organized by some classification into distinct levels, where there is a relationship between the levels. Either the higher levels have some kind of authority or control over lower levels, or decomposition occurs down the hierarchy (the higher-levels can be broken down into their lower-level sub-components).

Notes:

1. Within systems engineering, there are several hierarchical structures of specific interest:

• organizational/management structure hierarchies

• process hierarchies

• system specification hierarchies: The information relating to a system can be arranged in a hierarchy that stretches from the system name at the top hierarchical level, down through the different specification documents, to the individual attributes. For example: hierarchical tag names.

Example:

Hierarchical Tag 1. Hierarchical Tag 2. Local Tag:

*The tags serve to locate ‘Local Tag’ in some specification hierarchy.*

Type [Hierarchical \*083]: Adjective [Relationship].

## Historic Level "Historic Level " (Of Attributes); Concept \*084. April 20, 2002

A joint term for the two Benchmark level specifications of attributes, Past and Record.

*Related Concepts:*

*. \*084={\*<Past>. \*<Record>}.*

*Note this does not include Trend and Future, which are benchmarks related to the future and are not in themselves historic.*

## Host System: Concept \*348. July 31, 2001

A host system receives the evolutionary *delivery steps.*

## Icon Concept \*161 December 11, 2004

In Planguage, an icon is a symbol, keyed (Keyed Icon) or drawn (Drawn Icon), that represents a concept. All Icons are graphic or pictorial in nature – they should not use words or national languages. But they can be annotated by words or symbols. They refer specifically to defined concepts, such as those in the Planguage Glossary.

Related Concepts [Icon \*161]:

• Keyed Icon \*144

• Drawn Icon \*085

• Symbol \*161

Synonyms [Icon \*161]

• Plicon \*161 - © Tom Gilb 2004, a concatenation of Planguage Icon.

Type [Icon \*161]: Symbol Class.

## Idea Concept \*086 March 17, 2003

An ‘idea’ is an intellectual concept of any kind.

A plan, scheme, project, aim <- Webster’s Dictionary.

A plan, scheme, or method <- The American Heritage Dictionary.

Notes:

1. We are primarily interested in attribute ideas, design ideas and project delivery cycle (Evo step) ideas.

Related Concepts [Idea \*086]:

• Design Idea \*047

• Process Improvement Suggestion \*088

• Plan \*108

Type [Idea \*086]: Design Concept.

## Ideal Concept \*328 August 12, 2004

An ‘Ideal’ specification represents a hypothetical, and perhaps only theoretical, best, level for stakeholders, irrespective of considerations, such as cost, available technology, or culture.

An Ideal parameter specifies an ideal level, on a defined Scale, under specified conditions for a scalar attribute.

An Ideal level is *never* a *committed* *target* level. It is a *benchmark* level. We might want to move towards it, hold it out as motivation, but we do not expect to reach it in practice. It could conceivably be reached in the real world. In which case we may have to reset our ‘ideal’ notion towards a *new* level of ‘ideal. In this sense ideal differs from ‘perfection’, which by definition cannot be improved.

Ideal is is arguably classifiable as both target and benchmark, depending on how it is applied in practice. If it is used to compare current Goal and Stretch levels with the Ideal – then it is used as a benchmark. If it is seriously pursued in the long term, and possibly reached, even for limited cases, then it is a target class of requirement

Example:

Scale: Average distance in meters from average available parking place, or public transportation individual step off site, to nearest available office entrance.

Ideal [Walking Impaired in Wheelchair]: Less than 100 meters <- Employee Survey.

Ideal [Senior Sales Executives & Others often in and out of office]: 50 meters.

Ideal [Office Staff using Public Transport]: 200 meters <- City Planning Guidelines.

Past [People using Public Transport]: 1,000 meters, [Drivers Parking]: 300 meters.

Goal [New Office]: 100 meters, [New Office, People using Public Transport]: 200 meters.

Ideal specifications document sentiment, but Goal specifications declare the officially adopted target that seems to balance with all other practical, political, and economic considerations.

Notes:

1. The reason we have other target levels, like Goal and Budget, is because *they* must be established by considering all constraints (such as time, money and technology). Ideal is idealist, and doesn’t have to worry about such mundane things. It can be established *without regard or respect for* economics or technology.

“The role of an ideal is not so much to be reached, but to focus the allocated time and effort toward achieving it. If indeed an ideal is achieved, a new ‘improved’ ideal should take its place to guide thought and action.” <-KEENEY92, quoting Russell Ackoff (1978) The Art of Problem Solving.

Related Concepts [Ideal \*328]:

• Stretch \*404

• Target \*048

• Benchmark \*007

• Perfect \*648

Keyed Icon [Ideal \*328]: >++ “Explanation, more positive than Stretch ‘>+’.”

In context: ------>++------>

Type [Ideal \*328]: Parameter [Scalar], Benchmark, sometimes Target level.

Historical Note: ‘Ideal’ was specified by Tom Gilb in 1996 after reading Keeney’s remarks [KEENEY92, page 210 (7.4)].

Note [July 13 2004]: Ideal levels might be, but are not necessarily, levels of perfection.

## If Concept \*399 February 5, 2003

‘If’ is a logical operator parameter used in qualifiers to explicitly specify conditions.

Note: 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.

Examples:

Goal: Legal Constraint [USA, If Law 153 Passed]: 99.9%.

Goal: Quality Requirement [If Europe, If Product XYZ Announced]: 60%.

Synonyms [If \*399]:

• Where \*399

• IF \*399

Related Concepts [If \*399]:

• Qualifier \*124

• Condition \*024

Keyed Icon [If \*399]: ? “A question mark before a condition.”

The ‘?’ may be used before a condition, with a space between them. It should not be used after a condition since it would be ambiguous with uncertainty about the validity of the specification itself.

Examples:

Goal: Legal Constraint [? USA, ? Law 153 Passed]: 99.9%.

Goal: Quality Requirement [? Europe, ? Product XYZ Announced]: 60%.

Type [If \*399]: Logical Operator.

*Historical Note: Use of ‘Where’ as a synonym was suggested by Don Mills, New Zealand,* www.softed.com*.*

## Initiator Concept \*424. July 31, 2001

The Initiator is the person or instance who initiated a specification.

Type: Parameter, specification concept.

Usage: to identify the person or instance who originally initiated a particular specification; for any reason, such as keeping them informed, finding out more about why they did it – or what they specified; to determine if they still have any interest.

Rationale: knowing the initiator identifies a key stakeholder, gives you possibility of finding out the intent and the real deep reason behind the specification. It is a kind of Source and a kind of Authority.

Adaptability:

Initiator: Marketing. <- Market Plan 16 April 5.6.3

## Impact Estimate Concept \*433 June 6, 2003

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.

Abbreviation [Impact Estimate \*433]:

• Impact “Often ‘Impact’ is short for ‘Impact Estimate’. See also Impact \*087”

Keyed Icon [Impact Estimate \*433]: ->.#

*Note: the icon is made up of the impact symbol ‘->’ and the estimate symbol ‘*#’, *with the ‘.’ indicating that it is made up of two concepts.*

Related Concepts [Impact Estimate \*433]:

• Impact \*087

• Scale Impact \*403

• Percentage Impact \*306

• Side Effect \*273

Type [Impact Estimate \*433]: Scalar Design Metric.

## Impact Estimation Concept \*283 July 17, 2003

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

Description [Impact Estimation \*283]: Chapter 9, “Impact Estimation: How to understand strategies and design ideas.”

Acronym [Impact Estimation \*283]:

• IE

Keyed Icon [Impact Estimation \*283]: ->.# “Made up of the impact symbol and the estimate symbol. Note it is the same icon as for Impact Estimate \*433”

Type [Impact Estimation \*283]: Method, Process.

## Impact Estimation Table Concept \*638 July 11, 2003

An impact estimation table (IE table) is the main output specification of an Impact Estimation process. An IE table shows estimates or actual measurements of the effect of any set of designs (architecture, strategy, solution) on any set of requirements. The emphasis is on the impact of designs on the performance and resource targets.

Notes:

1. An IE table can be used for a variety of purposes. These include:

• to compare alternative design ideas

• to select a set of design ideas

• to determine how complete a design is (how completely a set of design ideas meets the set of requirements)

• to identify areas where new design ideas are needed

• to monitor progress of Evo steps towards the requirements

2. An IE table can be a powerful presentation aid. Care must be taken that the level of detail presented is appropriate to the audience.

3. An IE table presents a ‘snapshot’ of how well the set of scalar requirements are being met *at a given point in time* by the set of designs.

Acronyms [Impact Estimation Table \*638]:

• IE Table \*638

Related Concepts [Impact Estimation Table \*638]:

• Impact Estimation \*283

• Cell \*308

• Impact Estimate \*433

• Impact \*087

• Uncertainty \*310

• Credibility \*035

• Evidence \*063

Type [Impact Estimation Table \*638]: Specification [Design Analysis].

## Impacted By Concept \*412 January 24, 2003

‘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 ‘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 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:

Danger List [A]:

Impacted By: {Help Desk Capacity, User Motivation, User Training, Bug Frequency}.

3. ‘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 ‘Supports’ icon can be used, A ->> B. meaning A is **primarily** the way we intend to achieve the requirement/value B.

Related Concepts [Impacted By \*412]:

• Risk \*309

• Threat \*309

• Dependency \*189

• Impacts \*334

• Supports \*415

Keyed Icon [Impacted By \*412]: ->

Note: Used in the same way as for Impacts \*334. B is impacted by A is written A -> B.

Example:

A -> B “B is Impacted By A = A Impacts B.”

Type [Impacted By \*412]: Parameter [Relationship].

## Impact Concept \*087 July 23 2008

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.

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.

If an impact is estimated, it is an Impact Estimate \*433.

Related Concepts [Impact \*087]:

• Impact Estimate \*433

• Impacts \*334

Type [Impact \*087]: Noun, Scalar Metric Data.

Keyed Icon [Impact \*087]:

->@ or ->.@

This icon symbolizes Impact (->) on a target (@) which is what design ideas ‘do’.

## Impacts Concept \*334 February 5, 2003

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 [Impacts \*334]: ->

Note: this arrow icon is identical to ‘Until \*551’ but the Until icon is only valid in the numeric context ( ‘30 ->60’). 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 [Impacts \*334]:

• Supports \*415

• Impacted By \*412

Type [Impacts \*334]: Parameter [Relationship], Verb.

## Implementable Specification Concept \*294 January 16, 2003

Implementable Specification refers to written specification that translates into *real implemented* systems. Most all major specification defects are found in implementable specifications. But not all defects in implementable specifications are major.

By definition, major defects *are far less likely to* be found in ‘background’ specification. But they are possibly there depending on how individuals specify.

It is important to distinguish between these concepts of implementable and background. Authors/writers need to try to make the distinction visually for readers (for example, by using plain text for implementable specifications, and italics for background specification, or specific parameters like ‘Note:’).

Notes:

1. Implementable sections of a specification can be called ‘meat’ and, background sections can be called ‘fat’ {typically – but not necessarily - notes, comments, remarks, background, summary, introduction, references, rationale}.

2. Checkers in SQC should concentrate on carrying out rigorous checking, at optimum checking rates on the implementable specifications. This gives better efficiency in finding major defects

3. When the visual distinction is made, and it is clear what is background and not; then quality control checkers can more easily, and more certainly, decide which defects are major and which are not. If the defects are found in background they are most likely minor. If found in implementable specifications, they are possibly major.

4. Parameters are fairly clearly either implementable specifications (Scale, Goal) or background (Ambition, Gist, Past) if you think about it. But local judgment must always be applied to decide if a specification is potentially major or not.

**Implementation Concept \*683** 14 Aug. 2017 (Born)

An implementation is the real-life representation of an idea.

Notes:

1. This concept was added so we can document and track changes to real instances of a design or plan.
2. An implementation would normally refer to a plan or design which spawned it.
3. But there could be implementations with no known corresponding plans
4. I envisage adding this to the Needs and Means tool to track actual implementation, both plans for, and any instances of. TsG

## Implementation Cycle Concept \*123. January 2, 2003

An implementation cycle is part of a result cycle. It consists of the (Evo) development cycle, production cycle and delivery cycle.

Implementation means ‘taking a plan, or an idea, and turning it into reality’.

Notes:

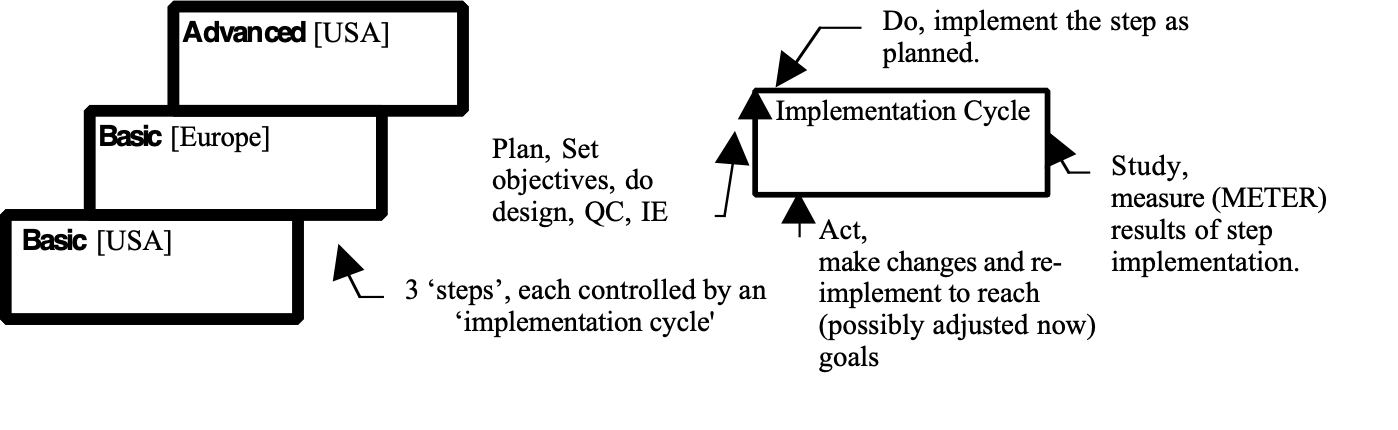
1. An implementation cycle is comparable to a part of Shewhart’s ‘Plan Do Study Act’ cycle (Statistical Process Control Method) ; the ‘Act’ part of it.

Note 28nov04 LB need to edit to wed’s def

2. The emphasis of an implementation cycle within Evo is on ‘contact with reality’ to obtain consequent feedback and enable later adjustment.

3. This can include factory production, software program writing, or any kind or realization of any design idea or prototype. It can include any kind of small change to a larger design.

4. The implementation cycle includes any related, supportive, development cycle process. The ‘development cycle’ is focused on research and development, rather than manufacture (‘the production cycle’) and distribution (‘the delivery cycle’) to customers, users and markets.

 *‘Implementation cycle’ contrasted with the ‘step’ concept.*

Synonym [Implementation Cycle \*123]: Implementation \*123.

Related Concepts [Implementation Cycle \*123]:

• Result Cycle \*122

• Development Cycle \*413

• Production Cycle \*407

• Delivery Cycle \*049

• Evo \*355

Type: Process.

**Implementation Instance Concept \*684** 14 Aug. 2017 (Born)

An implementation Instance is a particular the real-life representation of an idea. Possibly one of many planned or implemented or obsolte.

Notes:



## Improvement [of Process] noun: Concept \*088. July 31, 2001

An ‘Improvement’ is a suggestion for process improvement by SQC Checkers, or any others.

Synonym “Process Improvement Suggestion”, Process Improvement

Domain: CE.DPP.\*088

Keyed Icon: [\_\_\_].->.^[…] ---- decoded: [Improvement].->.^[Process]

(speculative suggestion July 31, 2001.TG. Idea.Impacts.Process)

Type: artifact of Defect Prevention Process (DPP).

*Note 1. Process Improvement suggestions are stimulated by current work with standards (procedures, rules, checklists, and forms).*

*Note 2. Process Improvement suggestions are usually reported and logged at an SQC Spec Meeting, a process meeting or anywhere else.*

*Note 3. The Improvements report is directed to the Owner of the process.*

*Note 4. Improvement suggestions need to be analyzed and tried out to find out if they have sufficient value-to-cost impact.*

## improvement "improvement" : (adjective) Concept \*009. July 31, 2001 "\*009.synonym.improvement"

Synonym for Benefit.

Note: not capitalized (‘i’).

"benefit "

Synonyms: relative improvement "relative improvement" (\*009), benefit (\*009), saving (\*429).

Related Concepts:

• \*??? Gap.

• \*429 saving

• \*009 benefit

• incremental impact +->

Keyed icon [\*009]: O ---- <==|--->

Explanation: a differential ‘gap’ (==) between some benchmark ( ‘-<-‘) and some target (‘|’) – Any target symbol can be used instead of the neutral ‘|’ symbol. (examples >, >>).

Note: incremental impact +->\*220

Type: {requirement type, attribute characteristic}.

“Improvement … is the attainment of a new level of performance that is superior to any previous level.” JURAN74, 16-1.

## Includes Concept \*391 June 5, 2003

‘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 sub-component 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 [Includes \*391]:

• Consists Of \*616

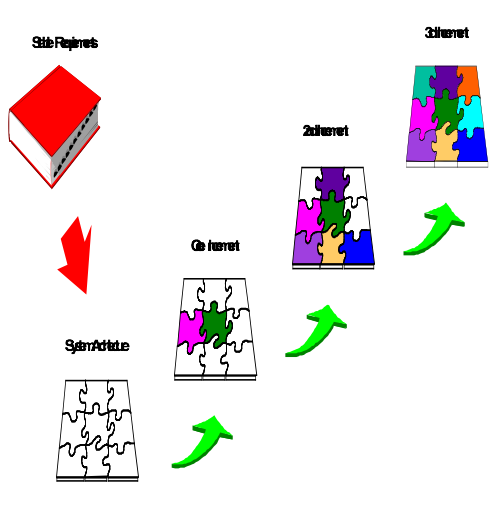
Keyed Icon [Includes \*391]: { }

“In context: X: {A, B} means X includes A and B.”

Type [Includes \*391]: Parameter [Relationship].

## Incremental Development Concept \*318 June 5, 2003

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

Figure \*318: Conceptual View of Incremental Project Management [HUMMEL02]. Compare with Figure \*355 in Evolutionary Project Management \*355

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 EVO02], among others, has taken the trouble to carefully distinguish these concepts!

Related Concepts [Incremental Development \*318]:

• Evolutionary Project Management (Evo) \*355

Type [Incremental Development \*

## Incremental Scale Impact: Increment. Concept \*307 August 8, 2005, 7June2019 [Icons]

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

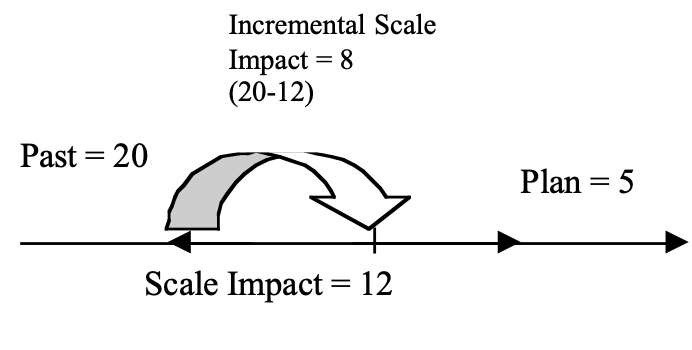
Example:

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).

Note: 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. Percentage Impact is used to convey this in an Impact Estimation table.

Keyed Icon: \*307 (Incremental Scale Impact)  **+.-> or ±.-> or** ∆ (delta)

**These keyed icons are used to represent the simple concept of ‘incrementation’**

**+.-> means ‘increase.on a scale’, ±.-> means ‘numeric change . On a scale’**

**“Δx means a (non-infinitesimal) change in x”.**

<- <https://en.wikipedia.org/wiki/List_of_mathematical_symbols>

‘In X’ we will interpret here as ‘in a Scale tagged ‘X’.

The ∆ symbol (keyed icon) is (2019) used in the [netplan.net](http://netplan.net) tool to indicate choice of an incremental numeric expression, versus an absolute numeric level on a scale.

Keyed Icon [Suggestion, \*307, Incremental Scale Impact, Aug8 05]

O---------<\_\_\_\_Increment X\_\_\_\_\_\_\_\_<|--------------->---------->

And

O----------<!\_\_Decrement Y\_\_\_\_\_\_\_\_\_\_\_\_<------------------>----------->

Explanation: (these keyed icons are useful for visualizing the degree of change, with Tags on participating objects, such as levels and designs.

1. the underline \_\_\_\_\_\_\_\_\_\_\_ represents the change from a defined benchmark. It can be texted to explain what it is.

2. The <| symbol (‘Increment Level’) represents a progress, from a defined benchmark. I t can be annotated like this. The leftmost ‘<‘ is the Past, before the increment. The right ‘<‘ is the Status after the incremental change.

O----<\_\_\_\_\_\_\_\_\_\_\_<| [Step3]------------>

3. the <! Symbol (’Decrement Level’) represents regression (bad direction compared to intent) related to a defined benchmark. It can be annotated like this:

O----------[French Connection]<!\_\_Decrement Y\_\_\_\_\_\_\_\_\_\_\_\_<------------------>----------->

4. Both levels are ‘Past’ once noted. That is why the ‘<’ symbol is used. The <| and <! Being modified Past symbols. The underline being the range \*552

O———-**<[Design A], ∆ [Design B]>\_**————-> Stakeholder Value X, Scale

***Figure 2.*** *Design A, reached about 50% of the way to our Target level. It is useful, but not sufficient. The incremental addition (∆) of Design B, allows us the reach the Target level, and beyond “\_\_\_\_\_\_\_>\_” (satisficing). The underline (\_\_\_) symbol expresses the visual degree of impact*

*'Design A, ∆ Design B’ is both useful, and sufficient.*

*(∆ is a Planguage Icon for ‘Increment’ Concept \*307)*

*Sufficient means ‘reaching or exceeding the Target level, on time’.*

*More tersely,* ***A∆B = >100%***

*(Design A, with Increment Design B, gives more than 100% of the Target requirement level, so A∆B is sufficient design. A∆B satisfices.*

*We can also express these ideas* numerically*:*

***X <- A 50%*** *Design A impacts Scale X by 50% (from Benchmark, which is implied 0%)*

***(X + ∆A) <- ∆ B 60%***  *Design B increments a further 60% (10% past the implied 100% level of the Target) on top of the increment we got from Design A (X ∆A).*

Illustration from a paper TSG worked in 070619 On a Grand Theory of design

Synonyms\*307 (Incremental Scale Impact)

\*307 Increment (noun, an increment) ∆

\*307 Incremental Change

Related Concepts \*307: Incremental Scale Impact

• Percentage Impact \*306 (Synonyms: %Impact, Incremental Percentage Impact)

• Scale Impact \*403

• Progress \*652

* Regress \*653
* Note 070619 I need to resolve use of ∆ to mean increment

|  |
| --- |
| Δ*x* means a (non-infinitesimal) change in *x*. |

Increment : *Mathematics* a small positive or negative change in a variable quantity or function.

Type: Numeric Variable.

Domain: Impact Estimation.

## Indicator "INDICATOR" : Concept \*195. July 25, 2004

An **indicator** is the result of a comparison of a measure or metric with a baseline quantity or expected result. <- Erik Simmons

A Planguage parameter defining an indicator type, and its condition: Example {red light, flashing}.

Synonym: ‘Sign’ \*195

Related Concepts [Indicator \*195]

• metric \*095

• measure \*132

Type: Planguage condition class.

*Example (Indicator as Parameter in Planguage):*

Goal [Clear] >90% <- IEEE Technical Interface Recommendation PL-967

Clear: Indicator {Any green or flashing panel light, any screen equivalent indicating OK}.

## Indirect Constraint:, Concept \*511 March 16, 2003

any *other specification, except a direct constraint,*  which indirectly has the effect of constraint.

This can include any other requirement {function, Goal, Wish, Stretch}, and previous and pre-emptive level of architecture or design, and any statement of priority attached to a specification (such as Authority, Source, Stakeholder).

Domain: Requirement.constraint.indirect.

Related Concepts:

• direct constraint 510. Any constraint which is specified in the format of a constraint and is intended to directly constrain performance, function, costs or design. For example Limit, Fail, Constraint, [qualifiers].

Type: Constraint

## Indirect Costs: Concept \*250. July 31, 2001

A resource (money, time, work-hours) needed to produce an input to a system.

Example:

work-hours are needed to produce high quality engineering drawings, which are themselves an input to a manufacturing and purchasing process. The cost of producing the engineering drawings to the required levels of quality is an indirect cost of manufacturing at a given level of efficiency.

## Information Concept \*320 March 12, 2003

Information is the interpretation an agent gives to data.

Notes:

1. Information includes {interpretation, conclusions, signals to act, basis for decisions}.

2. Planguage is both a way of organizing data (the language element organizes) to make it useful and meaningful, and also a set of processes (the defined processes, rules, and principles) to interpret data into useful information (such as Impact Estimation).

Information can tell us everything. It has all the answers. But they are answers to questions we have not asked, and which doubtless don’t even arise.

Jean Baudrillard (b. 1929), French semiologist. Cool Memories, Ch. 5 (1987; tr. 1990).

Related Concepts [Information \*320]:

• Agent \*163

• Data \*319: Raw facts before used by an agent to get information.

Type [Information \*320]: Communication Concept.

## Infotect: Concept \*568 July 12, 2002

The Infotect is

1. “Information systems architect.
2. Responsible for all aspects of the system – manual procedures, people, training, documentation, motivation – and if they are used at all, both hardware and software, including dataware).
3. Goal oriented: the infotect’s job is to design a way to reach the stated objectives, measurably.
4. Discipline synchronizer; design and plan control for all specialty engineers and planners.” <- [Gilb88, p351] Principles of Software Engineering management.

Related Concepts [Infotect, \*568]

• Systect \*565. an Infotect is a subordinate design function to systects.

• Software engineer, \*571. A software engineer is a parallel discipline subsidiary to systems engineering and systems architecture. A software engineer is primarily concerned with the process of providing instruction for computers.

• Systems engineer, \*574. A systems engineer could co-ordinate an Infotect if infotecture was a minor part of the system project. But it could be the other way around. An infotecture project for a corporation or organization could spawn supporting sub-projects which might require systems engineering for that subset.

• Softcrafter, \*573. A Softcrafter is usually called a programmer, and unfortunately also called a ‘software engineer’. We use softcrafter to describe the craft of writing, modifying or assembling computer programs. It is not an engineering or architectural discipline, no more than any other craft such as carpentry, electrical work, painter or plumber.

• Architect, \*193. an Infotect is an information system architect.

## Infotecture: Concept \*569 July 12, 2002

Infotecture is a discipline practiced by Infotects (\*568 see this). It is information systems architecture.

Related Concepts [\*569, Infotecture]

• Systecture, \*564. Systems architecture has, as a subset, ‘infotecture’, and software systems architecture (Software Systecture)

• Architecture ,\*192. Infotecture is a specialized architecture discipline limited to information systems of any kind, and not necessarily including computers or software!

## Infoware: Concept \*578. July 12, 2002

Infoware encompasses any *devices* for informing *people* about *ideas*. Infoware is information carriers in the widest sense of the word.

This includes anything people can read (test, diagrams, pictures) and any real artifacts which people can learn something from such as: models, prototypes, real systems, people, groups, organizations, any observable part of the universe.

Related Concepts [Infoware, \*578] – all these are types of Infoware

• Dataware, \*576

• Planware, \*577 Planware is plans (information, data) for humans .

• Documentation, \*579 documentation is primarily for human instruction about system operation

• Logicware \*575 is the ‘active’ logic directing computers in their actions. Logicware is not primarily Infoware – except to the degree humans access it for information directly. Some logicware is generated by computer software, for computer consumption and is not intended for human instruction – it is not Infoware.

• information \*320 is the interpretation we get from analyzing infoware.

## Inherited Concept \*516 May 4, 2003

A specification element is termed ‘inherited’ if it applies *implicitly* to a specific specification.

Inherited specification elements are not *directly* specified in a statement (that is, they are not local to a given specification), but they are inherited from related specifications, usually those at a higher level (a supra-specification).

Example:

Usability:

Type: Quality Requirement.

Kid Specifications: {Learning, Doing}. “Usability is defined here as consisting of these two qualities.”

Deadline: Time: End of Year.

Then in a different, non-local specification:

Learning: Scale: Time to <learn> System. Goal: 60 minutes.

Doing: Scale: Time to <correctly finish> defined [Average Tasks].

Goal: 5 minutes.

Goal [1st Release]: 10 minutes.

Type and Deadline are specification elements of the parent specification Usability. They are inherited by the Kid Specifications, Learning and Doing. The Goal [1st Release] does not inherit the Usability.Time deadline because it explicitly has another overriding specification.

Related Concepts [Inherited\*516]:

• Global \*373: Global specifications become inherited specifications where they apply.

• Local \*376: Local specifications at a high specification level can become global to lower specification levels and can therefore be inherited.

Type [Inherited \*516]: Specification Concept.

## Innovation Concept \*679 Dec. 2 2014

Order of magnitude, or better, improvement in performance/cost efficiency, in stakeholder-valued system attributes.

Notes:

1. The ‘order of magnitude (10:1)’ concept is an arbitrary, but useful, concept in the definition. It is our way of being somewhat more precise about the concept of ‘significant’ (improvement). In given situations we can certainly argue that far less ( 25%, 250% improvement) would be considered ‘innovation. So the degree of improvement needs to be argued in defined contexts. However, any ‘10 to 1’ improvement in the ratio of performance and costs, will almost invariably be considered true innovation. The less than this it is , the less ‘innovation’ degree.
2. Notice that we consciously avoided the trap of only considering the performance increases (and it can be several **concurrent** performance dimensions) alone. We believe that practical innovation will consider the **costs** (plural!) incurred, as an integral factor to be evaluated in considering the true degree of innovation. I.e. big performance improvements are interesting innovation, but doing so at half the price, rather than infinitely costly is ‘even more innovative’ in the real world.
3. Note that we very consciously include the notion of simultaneous improvement in any **useful set** of performance characteristics (for example Usability, and Security), together with any **useful set** of cost characteristics (for example Capital Cost, and Installation time). The more attributes improved, the more ‘innovation. All improvements deserve evaluation and credit.
4. So, our concept is ‘plural efficiency’ improvement, of significant dimension (10:1 as a benchmark idea, but not only possibility).

Thanks to R[aul.giurgiu@gmail.com](mailto:aul.giurgiu@gmail.com), Oslo for inviting me to discuss and define this concept, 1 Dec. 2014.

## Input Concept \*331 June 17, 2003

Input (or system input) is anything, which enters a system.

Notes:

1. Input has two primary uses in Planguage: it is either used to describe a function input (a resource attribute), or to describe a process input (a resource asset).

Synonyms [Input \*331]:

• System Input \*331

Related Concepts [Input \*331]:

• Process Input \*178

• Resource \*199

• Output \*179

Drawn Icon [Input \*331]:

For a function input (a resource attribute), the icon is an arrow entering the function icon (oval) from the left or west.

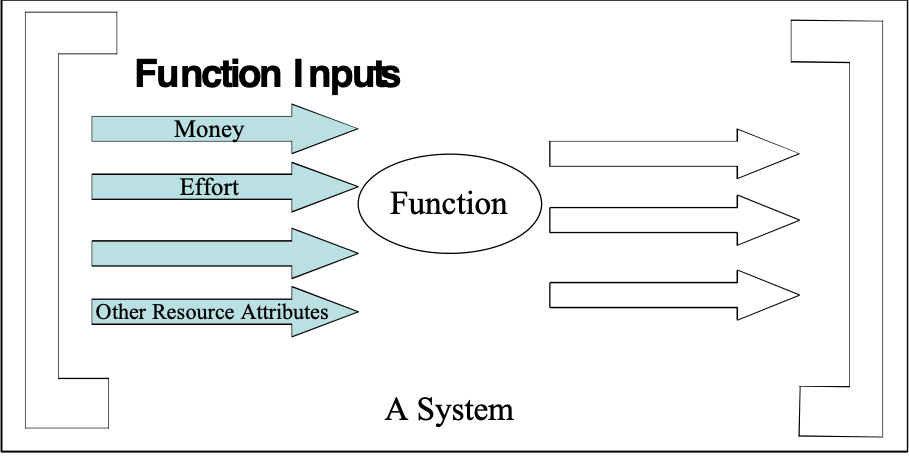


Figure \*331a: Function inputs, also known as resources (resource attributes)

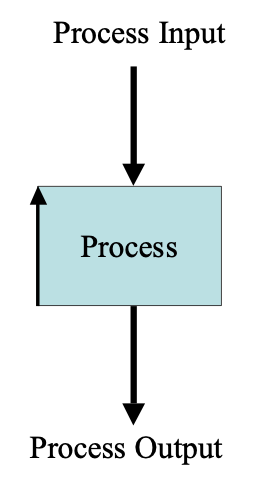
For a process input (a resource asset), the icon is an arrow entering the top side of the process rectangle.

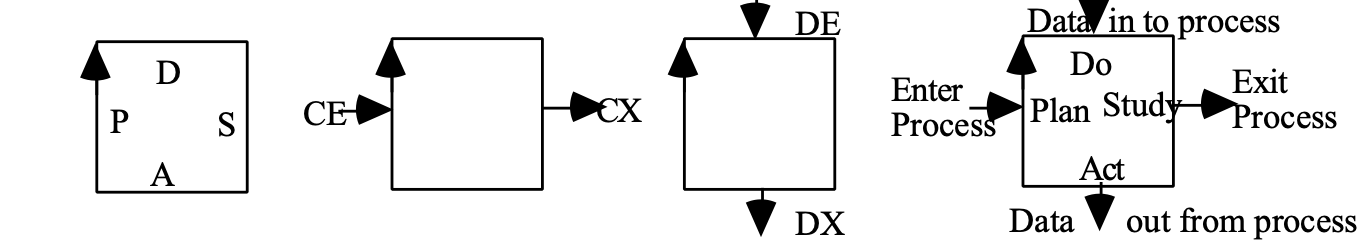
Figure \*331b: Process input

Type [Input \*331]: Resource Attribute or Resource Asset.

## Input/output axis. Concept \*260. August 1, 2001

The vertical axis of a process rectangle. The top represents inputs to the process (data and/or material). *DE below.* The bottom represents outputs from the process (data and/or material). *DX below.* Abbreviation: I/O Axis*.*

*Not co-incidentally the top side also represent the ‘Do‘ part of the PDSA cycle, and the bottom part represents the ‘Act‘ part of the cycle.*



## Inspection Concept \*051 November 21, 2002

‘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 [MAYS95].

Synonyms: Specification Quality Control (SQC) \*051, Peer Review (one type of, SEI CMM Level 3).

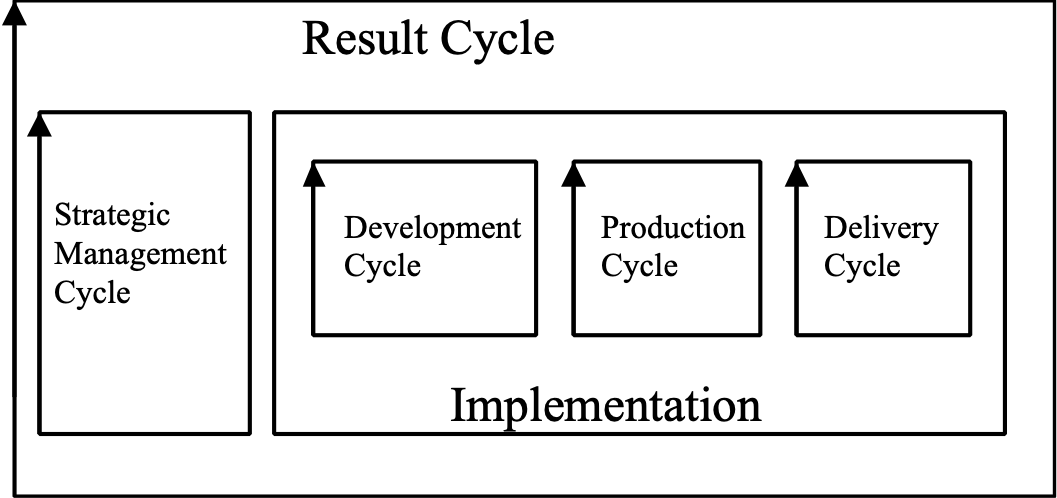
Type: Process.

Domain: SQC.

## Integration Concept \*357 February 24, 2003 tg

Integration is the process of inserting Evo step components into a system.

It is part of the production cycle (which is one of the three major sub-processes for implementation of an Evo step {development, production and delivery}).

Integration is part of the Production Cycle.

Integration works with developed or acquired step components, and does whatever it takes to integrate them into a system. Deployment within the delivery cycle then moves the system upgrades out to the result stakeholders.

Related Concepts [Integration \*357]:

• deployment \*358

• production cycle \*407

Type [Integration \*357]: Process.

Domain [Integration \*357]: Evo

## Intelligible Concept \*675 August 31 2014

Capable of being understood.

Notes:

Do we need a distinction between

1. a well articulated specification, that some people find difficult to understand
2. and a badly articulated specification that causes even less understanding

**Synonyms** [Intelligible \*675]

Understandable

Apprehensible

Graspable

Perceivable

Comprehensible

Comprehendible

**Antonym**

unintelligible

Related Concepts [Intelligible \*675]

## Interface Concept \*194 March 13, 2003

An interface is an agreed connection and boundary between system components.

Notes:

1. An interface can consist of anything useful: for example a device, a definition, or a language that helps to connect system components.

2. It is the device, or the means, which two system components use to connect with each other, for *any* purpose.

Related Concepts [Interface \*194]:

Type [Interface \*194]: System Component [Architecture].

## Internal Concept \*496 May 9, 2003

When describing the relationship amongst objects, internal is an adjective, which applies to any object inside the scope of a given object.

Examples:

• Internal stakeholders of a system

• Internal components of a device

• Internal modules of an application

• Internal staff within an organization

Notes:

1. ‘Internal’ is used with regards the ‘place’ scope of an object, rather than the ‘time’ or ‘event’ scope. ‘Where’ does an object stand with regard another object: within its extent of influence (internal) or outside it (external)?

2. An internal object can literally be a physical part of some larger object.

Related Concepts [Internal \*496]:

• External \*497

Type [Internal \*496]: Adjective.

## Internal Stakeholder \*494 Oct 27 2001

Internal stakeholders are stakeholders that directly impact a defined focus system. They are related to the environment systems supporting the focus system.

For example a tester is an internal stakeholder for a test process supporting the development of a product (defined as a focus system).

Related Concepts:

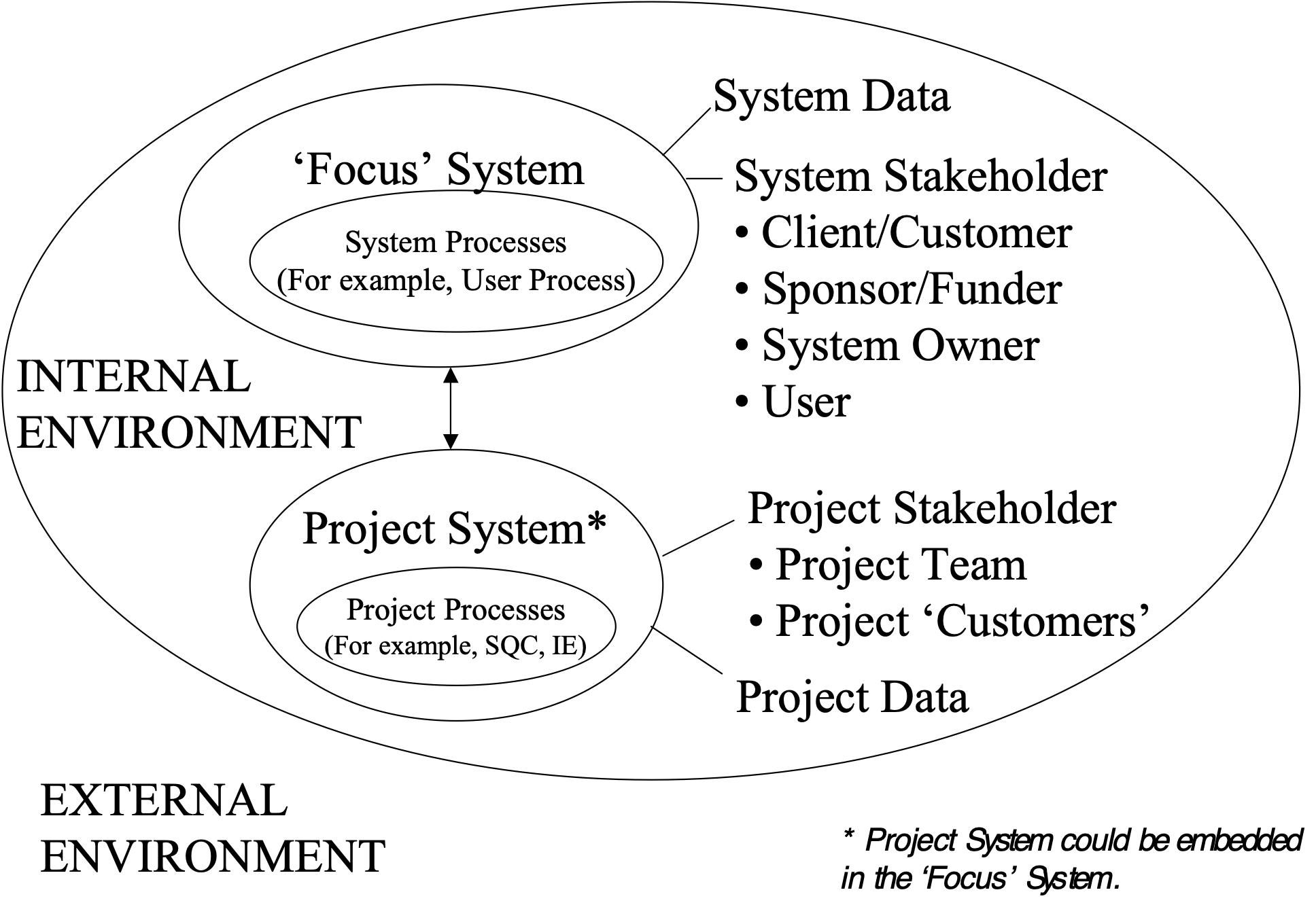
• Focus system \*492

• Stakeholder \*233

• External Stakeholders \*495

• Environment system \*493

*Type: relation concept.*

*Internal Environment diagram (suggested by L. Brodie 18 Nov 2002)*

## Is Part Of Concept \*621. November 29, 2002

A Parameter which indicates that a specification is a component or element of something else.

Related Concepts \*022

• Consists Of \*616

• Includes \*391 Includes can give a partial list of contents.

• Spawns/Generates \*392 Spawns implies active generation, not inclusion or consisting

• Component \*022

• Element \*022

*Keyed Icon\*621*

X {y}. meaning y is included in X

## Issue Concept \*276 October 20, 2004

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.

Example:

“An important distinction is that between ‘risks’ and ‘issues’. Issues can be defined as ‘previously unidentified risks/problems that are perceived to have occurred or are certainly going to occur, the probability of occurrence being 100%. Unlike risks, which may or may not come into play, issues will occur. It is more important to construct complete contingency plans for issues, therefore, since, with an issue, it is certain that an adverse impact on the programme will occur if no action is taken.”

<- Clive Fenton, [ELLIOT02, Chapter 2, Section 5]

*Notes:*

*1. In their 1970 paper Issues as Elements of Information Systems, Kunz and Rittel introduced Issue-Based Information Systems (IBIS) as a collaborative problem identification and solving tool. The concept is based on processes of argument. The system is particularly useful when at the outset a problem is either poorly defined, and/or preconceptions of the nature of the problem lead to participants in the discussion taking adversarial roles. To avoid the polarization that is common in debates, IBIS follows a procedure involving a decomposition of the problem. The key to the system is the idea of an issue, which is in effect a question. As the procedure takes place, particular kinds of knowledge artifacts are generated and recorded. Although finding a good solution to any given problem is by no means guaranteed, the process can reveal answers on which a consensus can be reached, or discover aspects of the problem to which other reasoning techniques may be applied. <- IBIS Vocabulary, http://ideagraph.net/xmlns/ibis/*

Related Concepts [Issue \*276]:

• Resolution: \*525

• Specification Issue \*529

Type [Issue \*276]: Parameter, Feedback.

## Item [SQC] Concept \*300 February 25, 2003

An item is a term for whatever gets reported by checkers and then logged by a Scribe during SQC logging.

Notes:

1. In the Specification Meeting an item is a member of the set of {issue {super-major, major, minor, new (as in found ‘new’ during a logging meeting)}, process improvement suggestion, question of intent to the author}.

Type [Item [SQC] \*300]: Specification Concept.

## Keyed Icon Concept \*144 February 25, 2003

A keyed icon is a non-alphabetic and non-mnemonic set of one or more key-able *characters that make up a keyed icon symbol,* with defined interpretation.

The choice of keyed characters is as closely related as possible to the corresponding *icon*, and ideally is acceptable to all language cultures.

The keyed icon will *normally* be *combined* with tags and other Planguage terms to give complete statements or expressions. For example ‘A <- B’ (B is source for A). A pure icon is a possibility, for example ‘O--->’ indicating a performance characteristic.

Some keyed icons, like many words, are ambiguous in isolation, but are unambiguous in context of other specifications.

*Rationale:*

1. *Keyed icons are intended to be shorter than any set of words for the concept*
2. *Keyed icons are intended to be independent of any particular language or culture*
3. *Keyed icons are intended to appeal to the graphical senses.*
4. *Keyed icons are intended to be useful for teaching and presentation of Planguage ideas and systems engineering concepts.*

[Editor Note to Publisher: Needs correct Appendix Reference Below.]

See Appendix ZZ, ‘Keyed Icons.’

Related Concepts [Keyed Icon \*144]:

• Icon \*085

• Icon \*161

Type [Keyed Icon \*144]: Symbol.

## Kid Concept \*266 February 23, 2003

The kid is the immediate *offspring* or subset of any component type with any sub-components (for example, function, process, plan, step or system).

Related Concepts [Kid \*266]:

• Parent \*267

• Supra \*264



Type [Kid \*266]: Role [Relationship].

## Kickoff [SQC]: "Kickoff " Concept \*286. August 1, 2001

Kickoff is an SQC sub-process. It usually involves a team meeting to agree on the master plan, and to commit to doing the planned SQC work.

Type: Process.

Domain: SQC

## Kin Concept \*353 November 13, 2002

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

For example, test plans, source code and user handbooks could all be derived from the same requirements or the same design.

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.

Type: Relationship Adjective.

Domain: SQC.

## Landing Zone Concept \*605 May 23, 2003 B

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.

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

Figure \*605a: A simple example showing multi-dimensional landing zones. It is landing within all the landing zones simultaneously that is the aim (Courtesy of Erik Simmons, Intel).

3. The multi-dimensionality 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 generally covers a success range and an acceptable range.

5. A landing zone may be tailored in specific instances to any numeric range desired. It does not have to use the generic default option of being in the success and acceptable ranges.



Figure \*605 b Diagram shows the extent of the landing zone for the two types

of scalar attribute, resource and performance

Related Concepts [Landing Zone \*605]:

• Range \*552

• Acceptable \*545, \*613

• Tolerable \*539

• Success \*548

Type [Landing Zone \*605]: Scalar Concept.

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

## Language: Concept \*373. August 2, 2001

An {agreed, understood, defined} {written, graphical or oral} means of expressing any ideas, concepts or notions.

We are particularly interested in a specialized language for systems engineering and management, which we call “Planguage” here.

## Level Concept \*337 March 5, 2003

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 [Level \*337]:

• Point \*337: A position on a Scale.

Related Concepts [Level \*337]:

• 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 [Level \*337]: |

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).

Type [Level \*337]: Scalar Concept.

## Level of Perception \*590 July 16, 2002

The ‘Level of Perception’ is the organizational or process level which a specification is at or related to.

For example, the architectural level is downstream of system level requirements level but is upstream of specialist engineering levels.

The level of perception may be specified in a qualifier: For example:

Goal [Architecture Level] 90%, [Test Level] 96%

Rationale: if the level of perception is undefined, misunderstood or unknown then many systems engineering concepts become undefined, such as what is a requirement and what is a design idea. Such things are relative to the levels of perception.

## Life Cycle Concept \*600 March 13, 2003 C

The life cycle is the entire duration of existence of a defined system – from inception to retirement or disposal, independently of the specific methods used for development and maintenance.

Note: a system life cycle is not to be confused with a development process. The life cycle concept applies irrespective of choice of development process. If an Evolutionary development process was chosen the slope of the curse might be different and the production/construction phases would be recurrent and would happen earlier.

Type [Life Cycle \*600]: Process Cycle.

## Limit Concept \*606 May 3, 2003

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 [Limit \*606]:

• Range \*552

• Fail \*098: Fail Limit

• Survival \*440: Survival Limit

• Catastrophe \*602: Catastrophe Limit

• Tolerable Limit/ Worst Acceptable Case \*539

Type [Limit \*606]: Scalar Concept.

## Local Concept \*376 January 11, 2003

‘Local’ is an adjective used to specify that something applies only to a restricted subset of a defined scope.

Example:

Risk: The budgeted human resources are not really available.

Note: This happened on the last 3 projects in this division. <-TG.

In this case the ‘Note’ is local to the above ‘Risk’ specification. This is because it is immediately below, preferably indented, and contains no other indication, such as a qualifier, which could indicate otherwise (Contrast to: “Note [All Software Projects] this happened ….<-TG.”

Example:

TF:

Type: Design.

Description [Telephone.Keypad]: Tactile Feedback.

Rationale: All Competitors have it.

The design idea TF is local to the component, Telephone.Keypad.

|  |  |
| --- | --- |
| Global  • Declares the scope of a component as being across everything within a stated scope  • Typically system-wide application | Local  • Declares the scope as being restricted to a specific subset within a defined, usually larger, scope  • Local implies more localized |
| Generic  • Declares something can be applied to components across a declared scope (usually quite wide)  • Multiple use for several different components is implied  • Tailoring to suit conditions of component subsets is possible (Features to cater for such tailoring might be in-built) | Specific  • Declares something can be applied only to the identified stated components within scope  • Restricted use under known conditions tends to apply. |

Table G.?: Four adjectives concerned with scope classification.

*For example: Within a requirement, a term may be defined locally by declaring a tag and stating a definition. It can then be used and reused (by referencing its tag) several times locally. The tag is not intended for reuse outside the scope of that requirement.*

Related Concepts [Local \*376]:

• Global \*375

Type: Adjective, Scope.

## Locus \*524 March 28, 2003

A ‘locus’ is the integration of all factors/elements that define a scalar level. A locus consists of:

• a specified scalar level

• a specified unit of measure

• a units of measure context.

The objective of all additional information is to improve the clarity and reality of the specification for purposes of getting control over it, and for contracting purposes.

The units of measure ‘context’ can be rich in any useful dimensions and any useful information about the scalar level.

The units of measure context primarily contains:

• place: geographic location

• space: any other spatial notions, ‘where’

• event: any other conditions, particularly events or conditions

The units of measure context secondarily contains:

• sources: where does the specification come from

• authority: what power is behind the specification

• priority: any information helping establish the priority of the specification

• any other information useful or necessary for understanding the specification

Time

Authority

Location

Component

Level

Scale

Ill. \*524 The scalar locus: all elements of information related to the level of specification are tied together in one related set. Any useful number of dimensions can be applied to the level.

The units of measure ‘context’ can arbitrarily be specified in the following ways:

• in the Scale specification itself; explicitly and as a Scale Qualifier.

• in a benchmark, target or constraint specification; explicitly and as a Scale Variable.

• as separate parameter statements attached to the level specification.

In summary, a simple specification of a number and a unit of measure is normally insufficient to clearly, completely and safely define a scalar level. A number of additional parameters need to be attached to the level, to bring out the exact meaning of the level. This set of attributes is the scalar locus.

Example:

Usability:

Scale: minutes for a defined [Staff] to master a defined [Task].

Goal [Staff =Doctor, Task = Circulation Analysis, Deadline = First Trial release, Market = USA] 60 minutes.

Authority: Regional Hospital Contract paragraph 6.3.2

Example \*524: the specified level of 60 minutes needs all the other elements of specification to build the complete set of understanding. This set is the locus of that specification.

Synonym [Locus \*524]:

• Scalar Level Locus \*524

• Scalar Locus \*524

Related Concepts [Locus \*524]:

• Units Context \*559

• Units of measure \*560

• Scale \*132

• Scale Qualifier \*381

• Scale Variable \*446

Type [Locus \*524]: Metrics Concept

## Logging Concept \*089 November 20, 2002

‘Logging’ is the writing of notes (logs) according to defined standards with the intention of their later use.

Notes:

1. Within SQC, logging occurs at two points: during specification meetings (when reported issues, process improvement suggestions, and questions of intent are logged), and during process meetings (when suggested process defect causes, and corresponding process improvement suggestions, are logged).

2. Informal voluntary note taking is not logging.

Type: Activity.

Domains: SQC, Systems Engineering.

## Logical Page Concept \*103 January 8, 2003

A logical page is defined as a defined number of *non-background* words.

Default Logical page volume: If no other definition is given, use ‘300 non-background words’ per logical page as default.

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

Note:

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

Abbreviation: LP.

Synonym: Page [in an SQC context].

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. This is not a Planguage defined term.

• Background \*507

Type: metric definition.

Domain: SQC.

## Logicware: Concept \*575 July 12, 2002, Edit Aug 19 2010

Any information which is intended to control the logical action of a computer.

This includes the obvious; computer programs (logic itself), including any data that in fact controls the computer logic, such as user parameters which control computer logic, and data integrated into the programmed logic itself. It would also include any logic at any level such as operating systems, software layers, and even hardware based firmware. It includes logic written by people, or generated by other logicware.

Related Concepts [Logicware, \*575]

• Dataware, \*576 dataware does not necessarily get used to control computer logic.

• Planware, \*577 Planware is plans for humans, not computers.

• Infoware, \*578 Infoware is information of any kind, a small subset of which might be Logicware.

• Documentation, \*579 documentation is primarily for human not computer consumption

## Logistics "logistics" : Concept \*242. "\*242.logistics" . August 2, 2001

The procurement, distribution, maintenance, and replacement of materiel and personnel.

<-American Heritage Dictionary.

Logistics "logistics" : \*242.Logistics "\*242.logistics" .

The procurement, distribution, maintenance, and replacement of materiel and personnel.

<-American Heritage Dictionary.

## Lower Level "lower level" : Concept \*090. August 2, 2001

Something lower in a hierarchy than high-levels.

Related Concepts:

• High-level (\*082) a position in a hierarchy of defined system elements, which is ‘high’ relative to a defined set of those elements.

Type: Relationship specification.

Domain: System.sub-system.level

## Major Defect Concept \*091 November 20, 2002

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: M *(Intentionally written with a capital M)*, Major.

Related Concepts:

• Defect (\*043)

• Minor Defect (\*096)

Type:

Domain: SQC.

## Malfunction Concept \*275 January 13, 2003

A system has a ‘malfunction’ when it does something not intended, or not desired, according to its design/construction specification, or according to reasonable stakeholder expectations (even though there may be no official specification about that expectation). *Malfunctions are undesired outputs (including no outputs) from a ‘real’ system.*

Error/Slip 🡪 (Specification Issue) 🡪 Specification Defect 🡪 Fault/Bug/System Defect 🡪 Malfunction

Notes:

1. Severity: a malfunction can have any level of severity or impact from catastrophically permanent destruction, to hardly noticeable variability of information or timing. It all depends on the nature of the specification, which is violated. Consequently, to really understand a malfunction, a notion of severity of impact needs to be added. This impact all depends on the environment impacted at the time of malfunction.

2. Human system stakeholders, users or operators may commit errors in connection with real systems (as opposed to specifications of those systems), which lead to system malfunction.

Classes [Malfunction]:

• System Malfunction: entire system malfunctions.

• Component Malfunction: a component of a system malfunctions.

Related Concepts [Malfunction \*275]:

• Error \*274

• Specification Issue \*529

• Specification Defect \*043

• Fault \*339

Type [Malfunction \*275]: Risk Analysis

## Master Definition Concept \*303 May 9, 2003

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.

Type [Master Definition \*303]: Parameter, Specification Concept.

## Master Term "master term " : Concept \*159. August 2, 2001

The primary, or only, reference tag to a master definition (\*303).

## Materials: Concept \*326. August 2, 2001

‘Materials’ are non-information process-inputs and process-outputs.

## Maximum. Concept \*537. March 30 2002

A Maximum specification

is a constraint citing the highest allowable numeric level of a scalar variable in the operational variations of a system.

Usage: the specification is used to define operational characteristics of a system after handover to stakeholders. It can be used to define one type of testing of a system. It may be accompanied by specifications of why (Justification, Source, Authority, Risk for example) and also some specification of the action the system is expected to take if it does fall below that level (Action).

Justification: we could have used another concept for example ‘Fail’ . But this concept makes a distinction between delivered levels (fail) and operational variations and process control of the operational variations of a system. A system could be delivered well above the Fail (or Survival) level but in operation exceed the Fail limit due to circumstances unknown or uncontrolled at the time of system testing and handover. This concept can be used in contracts to distinguish between handover levels and operational levels of performance or cost.

Type: Operational Constraint

Related Concepts:

• Fail limit \*098 specifies failure levels of a system

• Survival Limit \*440 specifies the extreme limit at which a system can survive

• Minimum \*536

• Catastrophe Limit \*602

Parameter: Maximum, Max (abbreviation)

Synonyms: Maximum Operational Level

Keyed Icon [Maximum, \*537] ^ (in context scale) example ----^---->

(the same symbol in another context indicates action ^[Process].

## Means (Parameter) Concept \*044. August 2, 2001

A parameter placed between a tag and a definition, equivalent to {a colon, Defined, Defined As} for readability.

*Bananas: Means: A fruit with a yellow peel , oblong, white colored fruit inside.*

*Synonym:s*

*• Defined (\*044)*

*• Defined As (\*044)*

*• Colon delimiter. Usually has same effect between a tag and a specification.*

*Type: Parameter.*

## Means (Concept): Concept \*047. August 2, 2001

The devices by which Ends are reached.

A Design.

Synonyms:

• Design Idea (\*047), see ‘Idea’ (\*086).

• Design (noun) (\*047) A design is a ‘specified, consciously selected, means’ to reach defined ‘ends’.

• Solution (\*047)

• Strategy. (\*047)

*• Process Improvement Suggestion (\*088)*

• Architecture (noun) (\*192). The highest level of design.

*• Means Objectives (\*215): Objectives , which if attained, serve to help us reach Strategic Objectives.*

*Related Concepts:*

*• Ends: {Goals Objectives, Aims, Targets, Values}: The purpose of using the ‘means’.*

Type: stakeholder value, goal specification concept

Domain: problem solving. Including engineering, architecture, strategic planning.

Rationale: very general and short word which captures the notion in a general way, without necessitating a more-detailed term (like architecture or design). Well understood word by most people.

Keyed Icon [Means, \*047]: […] [Means-Tag]

(keyed icon , square brackets symbolizing the rectangle, contains Means tag, underlined).

*Note 1. Means can include any means whatsoever, which arguably contribute to reaching defined end state objectives. In particular they can include 'Means Objectives' (\*215), not just Strategies, Design Ideas).*

“Perfection of means,

And

Confusion of ends,

Seems to characterize our age”

Albert Einstein

Http://albert.bu.edu

## Means Objective "Means Objectives" : Concept \*215 August 2, 2001/April 1, 2003 (added moe mop), 2019 + \*047

Means objectives are a ‘strategy’, in the format of a performance objective, to meet strategic objectives. They exist only to support meeting strategic objectives.

Note 1. As viewed from the point of view of a defined stakeholder, means objectives are intended *merely* as a way to reach the higher-priority strategic objectives of that stakeholder.

Note 2. Means Objectives are always relative to a defined level of engineering or management. Your means objectives can become your subordinate’s strategic objectives.

Note 3. The classification ‘means objectives’ is a temporary point of view, not a permanent classification for all points of view.

Rationale: The usefulness of the means objectives concept is to help us keep sight of the objectives we are responsible for (our strategic objectives). Means objectives are secondary, and must not stand in the way of meeting our strategic objectives. So, by classifying something as ‘means objectives’ we recognize that these objectives are not fixed or ‘holy’. They can and should be changed or replaced, as soon as we believe that it would better serve the interests of meeting the strategic objectives. The means objectives are also recognized as being of an equal status and priority to any other strategy specifications, which also are serving the purpose of helping us to reach our strategic objectives.

Related Concepts.

• Strategic Objectives \*214 : the objectives we are responsible for. The ones we hope to serve by reaching the means objectives targets.

• Strategies: a means objective is a strategy. It is different from other strategies in that it is formulated in terms of performance targets. To reach these performance targets we will need specific strategies.

• Measure of Effectiveness \*486

• Measure of Performance \*482

“With a means objective, the answer to the question ‘Why is it important?’ is always an end that follows from that means.” <- [KEENEY92, page 78]

Keyed Icon [Means Objective, \*215]: ->.@ (Impacting.Objective).

Any requirements or objectives icons can be used to represent a means objective.

Synonym: Means Requirement.

Related Concepts:

Design Idea \*047 (the subtle difference being that the Means Objective is always a variable objective)

Type: Requirements hierarchy class.

Domain: Requirements.Solutions.Means Objectives

The term ‘means objective’ was coined by Keeney [KEENEY92].

## Means Constraints: Concept \*428 August 2, 2001

Means Constraints are *constraints imposed at levels ‘below’ and supporting us. This means that we can therefore overrule means constraints, if we need to in order to better satisfy our strategic requirements.*

Note 1. Means Constraints are at the same level (‘below’ us) as are Means Objectives (which are Performance requirements).

Related Concepts:

• Fundamental Constraints, are above us and outside of our control, givens.

• Strategic Constraints: are self imposed in our planning process, and can be changed by us.

These are different levels and viewpoints of the same class (constraint) of ideas.

Type: requirement .

Domain: Requirements.Constraints.Means.

Rationale: if we know about these constraints, and classify them as means constraints, then we are aware of their priority as being lower. We can make tradeoff decisions with this knowledge.

Example:

Effort [Tele]:

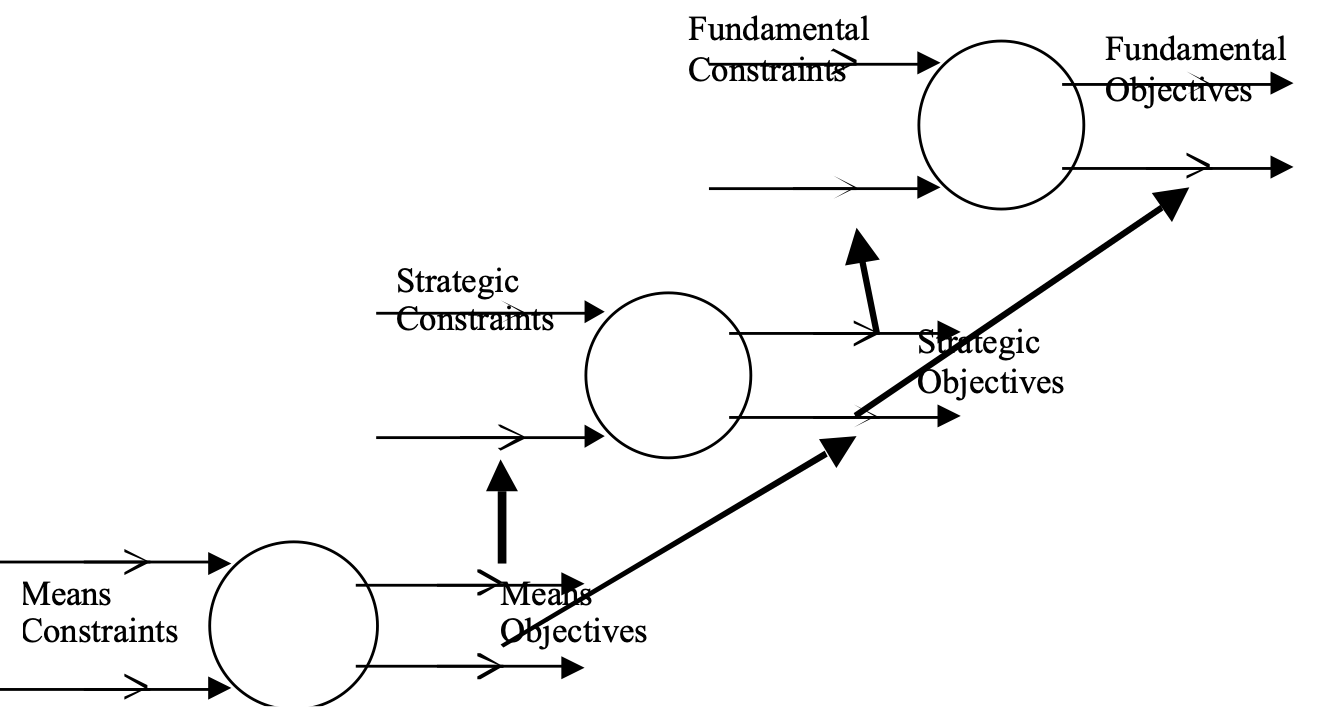
Scale: Engineering Hours.

Limit [ Department Tele, 1st whole Year of Project] 10,000 Eng. Hours <- Tele dept. Engineering Budget.

Type: Means Constraint [Project XYZ].

Authority: Tele Department Director.

Note that the resource constraint is identified as a means constraint in relation to a specific project.

**

Means Constraints are at a systems level below or before our own primary area.

## Measure (noun) Concept \*132. July 30, 2004

A measure establishes the “extent, dimensions, capacity, quality or performance level, etc. of anything, especially as determined by a standard”

Adapted from The Handbook of Software Quality Assurance, 3rd Ed., by Schulmeyer and McManus, PTR 1998, by Erik Simmons, Intel. Modified by TG July 30 04 to include quality or performance level

Notice the one dimensional nature of ‘measure’ compared to the related term ‘metric’, which can be complex (several measures).

Synonym [Measure, \*132]

• the Planguage parameter Scale

Related Concepts [Measure, \*132]

• Metric \*095

"Scale"

## Measure, To Concept \*386 May 9, 2003

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 future results based on past data.

Related Concepts [Measure, To \*386]:• Scale \*132

• Meter \*093

• Quantify, To \*385

• Estimate, To \*059

Type [Measure, To \*386]: Verb.

## Measure of Effectiveness Concept \*486 September 11, 2001

A measure of effectiveness is the quantified impact of a defined measure of performance (\*482, MoP) on a defined system under defined environmental conditions.

Abbreviation: MoE, MOE.

Example: if the Ease of Learning Task X impacts Cost of Training, then the Cost of Training is a measure of effectiveness of the Ease of Learning attribute. The Ease of Learning itself is a measure of performance (\*482).

Related Concepts:

• Means Objectives \*215 Means objectives are MoPs which impact Strategic Objectives (MoEs in relation to means objectives).

• Strategic Objective \*214. Strategic Objectives are the measures of performance we seek by reaching the target levels of a means objective (a Measure of performance in relation to the strategic objective).

Type [Measure of Effectiveness \*486]: Metric concept

## Measure of Performance. Concept \*482 September 11, 2001

A Measure of Performance is a quantitative expression of performance or effectiveness attributes of a system.

The measure of performance expresses the isolated measurable performance of a system, without respect to its knock on effects (measures of effectiveness \*486) in other systems.

Synonym:

Related Concepts:

• Performance \*434

• Scale (Measure) \*132

• Effectiveness. \*053

• Measure of Effectiveness (MOE) \*486: the MOE is the resulting effect in another system, and defined environment of an impacting measure of performance.

Abbreviation: MOP, MoP

See SPROELS02

Type [Measure of Performance \*482]: Metric concept

## Mechanism: Concept \*527 Feb 24 2002

A system of mutually adapted parts working together as ... a means by which a particular effect is produced. <-OED

A mechanism is HOW the system does a function ("means").

Providing the service is WHY a system does a function

\* The function is WHAT the function does

\* A mechanism is HOW the system does a function ("means")

\* A process is (part of) HOW the mechanism does a function ("activity"). <- Don Mills, NZ, 2002

## Mega-step: Concept \*345. August 2, 2001

An evolutionary ‘step’, or step plan, which includes two or more delivery steps.

Note 1.This includes mega-steps that include other mega-steps, which ultimately contain, or are intended to contain, delivery steps.

Synonym: Complex Step.

Type: Project unit-of-work concept.

Domain: CE.Evo.Step.Mega.

Related Concept:

• Elementary Steps. Evo steps which are not decomposed into other steps.

## *Meter -|?|- Concept \*093 May 5, 2004*

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” [DEMING86]

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, Euro]: 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, 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.

Notes: on the relationship between Meter specifications and Scale specifications:

Notes  
1: All Meters are based on whatever their corresponding Scale says.  
2. The Scale decides ‘what’ we are measuring. The Meter decides ‘how’ we will measure on that scale.  
3. The purpose of a Scale is to give meaning to numbers on that scale. To decide what is good and bad.  
4. The purpose of a Meter is to give us feedback about activity (variation) on that scale for a real system.  
  
  
We need to include the notion of Scale Qualifiers, and Target Qualifiers – in order to compare with Meter parameters and conditions.

Example:  
  
Call Help Speed:  
  
Scale: average time needed for defined [User Types] to successfully complete defined [Tasks] under defined [Conditions].

Meter [User Type=Novice, Task = Help Caller, Conditions = Lingual Challenge]  at least 20 representative call situations with at least 3 extremes of Novice, Lingual Challenge, and Tasks

Goal [User Type=Novice, Task = Help Caller, Conditions = Lingual Challenge]  10 minutes.  
  
Lingual Challenge: defined as: the caller has substantial problems explaining or articulating the problem and situation, due to any cause, such as non-native linguist, non domain expert, noise or disturbances.

From this example we can see that:

1. Scale determines what we are exercising control over.
2. Meter determines the accuracy/quality/reliability/credibility  and cost of any feedback from the real system.
3. The meter can be varied at any time to improve the Quality/Cost ratio of feedback, but the essence of what you are trying to control is defined in the Scale, including real environmental conditions present in the long term operating environment.

Conclusion:  
1. If you are trying to get control of some aspect of a system then all information about that aspect needs to be put into the Scale (or at least in the ‘Scale + Target [Qualifier] ‘combination, to be more precise.  
2. In particular, if you want control over a real system, you should not rely on the test/Meter specifications themselves to give that control. The designer and architect are not looking at the Meter specifications. They are looking at the Scale +Target information.  
  
  
  
SO GUIDELINES, PRINCIPLES

1. If you want control over a variable attribute, you must state exactly what you want control over in the Scale +Target  area; do not put such specs in the Meter.
2. The Meter is there to give us sufficient feedback, at timely intervals,  at lowest const consistent with the quality of measurement we want to have, so that we can proceed with the engineering process or operations with the confidence we feel we need (a matter of risk control).

Keyed Icon [Meter \*093]: -|?|-

“The explanation of the icon is that it is a ‘?’ on top of a Scale icon, -|-|-.”

Synonyms[Meter \*093]: Measurement Process ( useful for beginners in the method)., Test Process (useful for technical people). Accounting Process (useful for financial people like CFOs).

Type: parameter, metrics concept.

## Meter (noun) "meter" : Concept \*094. August 2, 2001

A meter is anything defined with the Meter parameter.

## Method Concept \*547 May 3, 2003

A method is a body of systematic techniques used by a specific discipline.

Related Concepts [Method \*547]:

• Process \*113

• Planguage \*030: Planguage is a specification language and a set of methods.

Type [Method \*547]: Tool.

## Metric Concept \*095 July 30, 2004

A system of related measures that facilitates the quantification of an attribute.

The metric may be composed of one or more measure dimensions or characteristics.

For example ‘Mean time between failure in operation’ – has 3 defining concepts – ‘time’, ‘failure’ and ‘in operation’. This includes one ‘measure’ (time).

A *complex* attribute specification, meaning one defined with a set of several scales of measure, is *also* a metric expression. There is no implication that a metric is ‘elementary’ (that it has only a *single* Scale).

A ‘Scale (of Measure)’ parameter, or a set of them, is the way we specify metrics in Planguage.

Notes:

1. Metrics are used to express ‘concepts of variability’ clearly – in particular more clearly than mere words [GILB76].

Examples (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.’

Examples (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 and 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 [Metric \*095]:

• Scale \*132

• Meter \*094

• Measure \*132

• Level \*337

Keyed Icon [Metric \*095]: -|-|-> “Scale with units” or ---> “Scalar Arrow, a simplified keyed icon without *explicit* units”.

Several other defined keyed icons can also be used to express a metric (For example, Fail and Goal: O-----!---->---->).

Type [Metric \*095]: Numeric Value.

## Milestone: Concept \*372. August 2, 2001

A milestone is any defined measure of project progress.

In Evolutionary Project Management the Evo steps successful completion are the normal milestone concept. But any set of Evo steps (Mega-steps) can be used to define interesting milestones. See example below.

Type: Planguage Parameter.

Domain: Project Management.Process Control. Metrics. Milestone.

Keyed Icon: ----|----- (1st draft, not determined) or ^[Process]->Exit

*First Release:* ***Milestone****:*

*{Steps: {Internet Option, Card Privacy },*

*Committee Approved [USA, Europe],*

*Product Orderable On Website}.*

Example of defining a Milestone as a set of conditions.

*Case [Microsoft]Microsoft used a large incremental step of about 25% to 33% of a project, and of 6-10 weeks duration as what they called a milestone. The milestone step contains everything necessary of components and quality to, in theory, deliver to customers. Early milestones contain the most critical components[Cusumano95]. We consider the Microsoft milestones to a type of evolutionary mega-step.*

## Minimum: Concept \*536. March 30 2002

A Minimum is a constraint specification citing the lowest allowable level of a scalar variable in the operational variations of a system.

Usage: the specification is used to define operational characteristics of a system after handover to stakeholders. It can be used to define one type of testing of a system. It may be accompanied by specifications of why (Justification, Source, Authority, Risk for example) and also some specification of the action the system is expected to take if it does fall below that level (Action).

Type: Operational Constraint

Related Concepts:

• Fail Limit \*098

• Survival Limit \*440

• Catastrophe Limit \*602.

• Maximum \*537

Parameter: Minimum, Min (abbreviation)

Synonyms: Minimum Operational Level

Keyed Icon [Minimum, \*536] \_ Underline example ----\_---->

Justification: we could have used another concept for example ‘Fail’ . But this concept makes a distinction between delivered levels (fail) and operational variations and process control of the operational variations of a system. A system could be delivered well below the Fail ( or Catastrophe) level but in operation exceed the constraint levels due to circumstances unknown or uncontrolled at the time of system testing and handover. This concept can be used in contracts to distinguish between handover levels and operational levels of performance or cost.

## Minor Defect Concept \*096 November 20, 2002

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.

Abbreviation: m, minor

Related Concept: Major Defect (\*091).

Type:

Domain: SQC.

## Mission Concept \*097 March 19, 2003

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 un-dramatic statement of the main function of a business or organization.

Mission, like function, intentionally excludes specific levels of attributes in its description.

Examples:

Mission: We make semiconductors.

Mission: We provide business solutions in manufacturing software.

Related Concepts [Mission \*097]:

• Function \*069

• Core Ideology "controllable" . Concept \*664

• Core Purpose \*662 The ‘Why’ (Mission is ‘What’)

• Scope \*419 (Mission is one type of scope delineator)

Type [Mission \*097]: Function Requirement.

## Mitigation Concept \*670 August 14, 20 2014 Version 15:57

Mitigation, is any Strategy which is intended to deal with a Threat (potential Attack), or an actual Attack, within the system, or by being added on after the Threat has successfully penetrated the system (Attack has occurred).

Built-in Mitigation: are Strategies that are an inherent part of a defined system, and are intended to cope with defined Threats.

It is essentially a ‘security’ design to cope with the threat (at type of Attack). But in many cases the concept Mitigation covers a broader range of systems ideas

Post Damage Mitigation: are mitigation strategies that are added to the system, after a Threat has penetrated the Built-in Mitigation, and caused negative consequences.

It may or may not have been pre-planned. It may recover from consequences (like a system restart or recovery) , or recover loss of resources (like a contract fine)

Planned Mitigation:

Is formally planned Mitigation in advance of an Attack

Classes of Mitigation Planning

Threat Mitigation:

Strategies that reduce the existence of certain threats in type and volume

Attack Mitigation:

Strategies that reduce the existence of certain Attacks in type and volume

Post Attack Mitigation

Strategies that reduce the Damages of successful attacks to a system

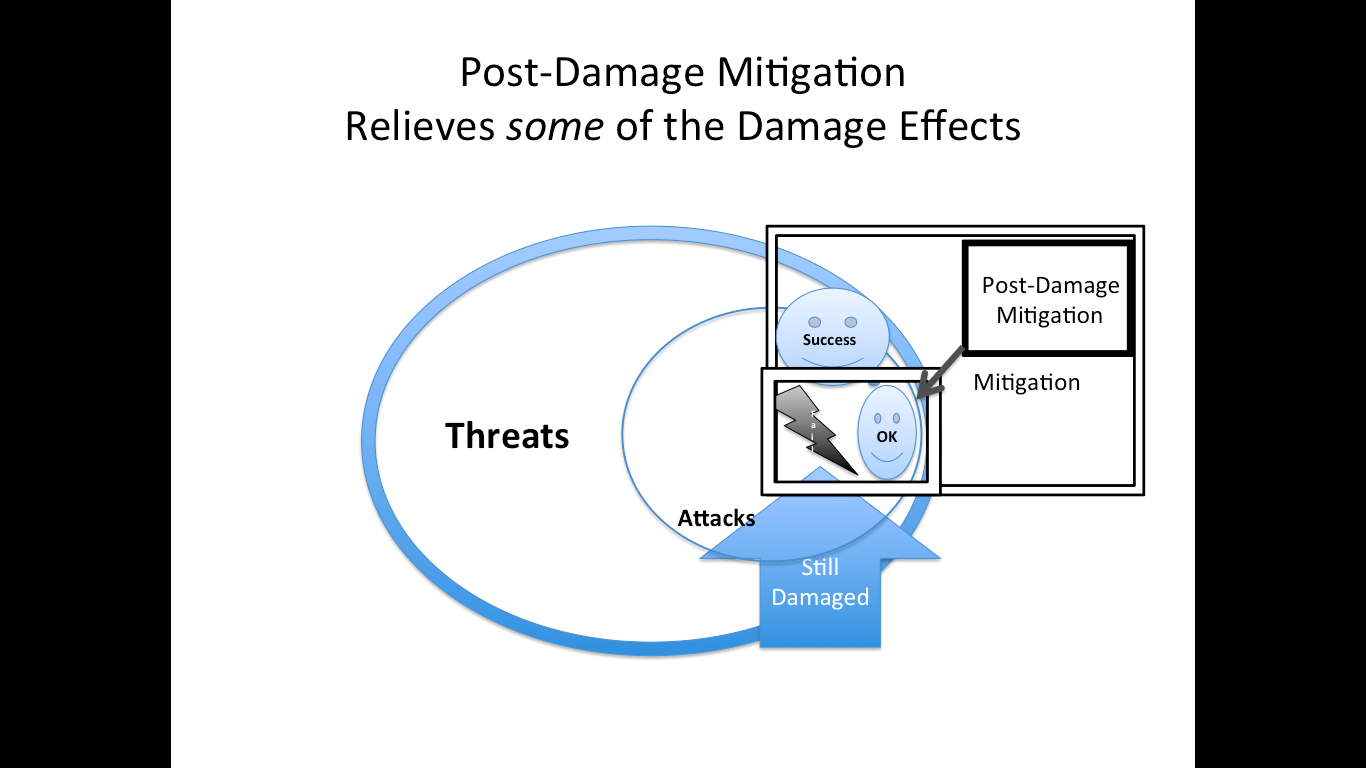


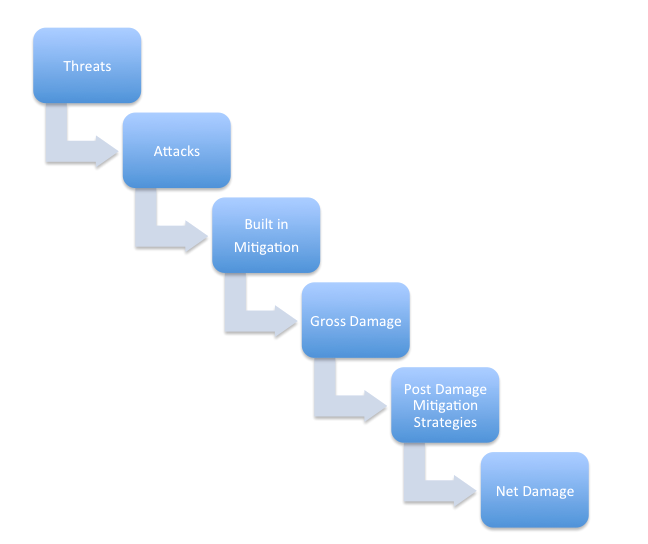
Diagram \*670: Threats are potential attacks on a system. Mitigation is intended to ‘deal with’ both threats (reduction, elimination) and real Attacks (detection, stop, contain, destroy, fix before damage). Some ‘built-in to the system’ mitigation will fail to deal with attacks successfully. Damage will occur, of some sort. Post-damage mitigation will try to alleviate the damage to some degree (‘OK’). And to some degree it will fail, and the damage effects will remain (Still damaged).

…….

**Notes**:…..

*ORIGIN late Middle English: from Old French, or from Latin mitigatio(n-), from the verb mitigare ‘alleviate’*

*I have on the spur of the moment concocted the terms Built-in mitigation, to describe mitigation consciously planned, and built into a system (for example devices for detecting attacks, containing attacks, thwarting attacks, and compensating for attacks); and Post Damage Mitigation to cover strategies perhaps planned, in the event of damage, but not built, or applied until the damage was done (example insurance payouts). It seemed, from a planning point of view these were interesting distinctions to make. TG 140814*



**Related Concepts**:

• Risk \*309

Threat \*671

Attack \*672

**Synonym**:

Security (not defined in Glossary, is defined in CE 5)

Negative Results

**Type**: Parameter

## Model. Concept \*448 January 6 2008

A model is an artificial representation of an idea or a product. The representation can be in any useful format including specifications, drawings and physical representations.

Rationale: to convey information about an idea without having to deal with a full scale and real version of it.

Notes:

a. A given model may have any interesting limited Function (what the model does)

b. and any given Design (how the model achieves its performance and cost characteristics - its efficiency)

c. a model can have any real set of perceivable (measurable, testable) Performance characteristics

d. a model can have any real set of Costs (for acquisition, learning, setup, operation, evaluation)

e. a model can have a set of constraints (legal, cultural, conventional, physics)

Related Concept: Template , \*254

• templates purpose is to give us ideas about possible patterns or ideas.

• model purpose is to convey information about one or more aspects of a system or product.

Type: engineering concept.

## Mode. Concept \*485. September 5, 2001 tg 1653

A mode is a defined type of operation of a system.

Related Concepts:

• State: \*024. A mode (of operation) can be described by may variable, possible, and impossible states.

Synonym:

• Mode of operation \*485

• Functional Mode \*485.

Type: system analysis viewpoint

## Motivating (adjective) Concept \*443 July 18, 2002

A ‘motivating’ systems component, is one which causes people to act.

Definition [Formally] motivating defined [Systems Component] is a cause of defined [Stakeholders] for doing defined [Actions].

Usage 1. Motivating Needs: the stakeholder needs which cause them to fund or otherwise support a project or any effort to achieve those needs.

Rationale: if a systems engineer can correctly identify the motivating components, they can derive relevant requirements and designs; and as a result get better funding and support for their efforts.

Type: system component attribute

“By not aligning measurements and rewards, you often get what you’re not looking for.” Jack Welch, p 387 [Welch01].

## Must Avoid (Parameter): \*098 . August 19, 2002

See Fail (-ure) Limit \*098 preferred use.

Must Avoid is a parameter, an abbreviation for the concept ‘Fail limit’ (\*098). It is being kept in the glossary as a synonym for what we would now recommend and prefer as a parameter for the concept: ‘Fail limit’, and the parameter ‘Fail’..

Must Avoid is defined as a level that Must be avoided if possible because there is some type and degree of failure at that level and at numerically worse levels.

Keyed Icon [\*098, Must Avoid] ‘>>’. (Explanation: double Goal level)

Synonym: Fail Level (\*098), Failure level (\*098), Must Avoid Level (\*098).

Parameter Abbreviation: Must (not recommended because of ambiguity and lack of precision).

Type: Scalar Constraint.

Rationale:

• to keep compatibility with the long term use of Must in Planguage. (in spite of a shift to preferred term Failure Level.

• to define a point at the beginning of a range of an attribute scale which must be avoided.

Note: the main reason we are keeping Must here is that we have used the concept since 1988. Some literature and some businesses have invested in teaching it, so we want to allow them to avoid optimization to a new term. It replaced ‘Worst’ : ‘Worst Acceptable Case’ before that). We do believe that the Failure Level concept is a more direct description of the concept, and that the Fail icon ‘!’

(----!-->---> for example) is a better icon than >> for a warning limit like Must Avoid, or Failure Point.. Certainly it is a general principle of Planguage that people can choose any term they like, to describe a concept.

Parameter: Must Avoid

Related Concepts:

• Fail limit \*098 synonym

• Must Do \*539 the level just slightly better than the Must Avoid level.

## Must Do. Concept \*539 May 29, 2003

Synonym of ‘Tolerable’ \*539

## Need Concept \*599 March 17, 2003

A ‘need’ is something desired by a defined stakeholder.

The implication is that satisfying that need would have some value for some stakeholder.

A need might not be agreed as a formal requirement, and it might not be prioritized such that it is actually acted upon (designed and implemented).

It is a term often used as a stakeholder view of a problem before requirements specification is carried out.

Related Concepts [Need \*599]:

• Problem Definition \*598

• Requirement \*026

• Design \*047

Type [Need \*599]: Problem Definition Concept.

## Non-Commentary Concept \*294 July 17, 2003

‘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’.

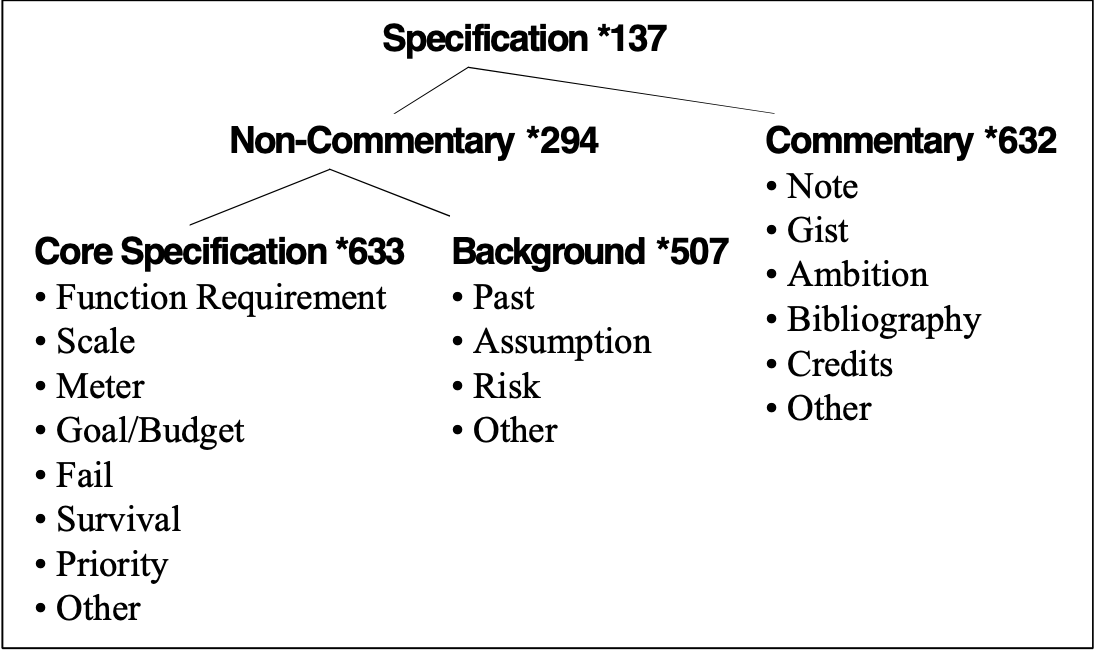


Figure \*294: The diagram shows the relationship amongst the different categories of specification.

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’ {notes (like footnotes), comments

(“like this”), remarks (Note), introduction, 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 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 [Non-Commentary \*294]:

• Commentary \*632

• Background \*507

• Core Specification \*633

• Specification \*137

• Major Defect \*091

Type [Non-Commentary \*294]: Specification.

## Note Concept \*018 May 26, 2003 E

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]. *“Source is background, but good for credibility and SQC.”*

*Source: The Encyclopedia.*

Synonyms [Note \*018]

• Notes \*018

• Comment \*018

• Remark \*018

Related Concepts [Note \*018]:

• Commentary \*632

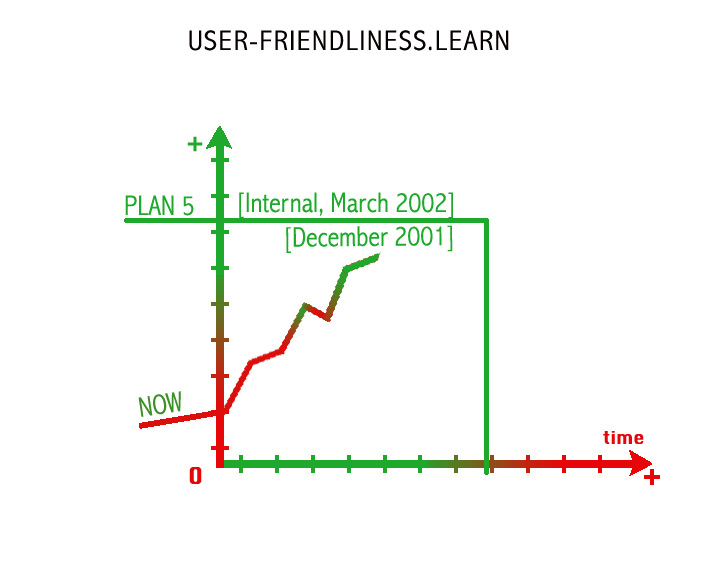
Keyed Icon [Note \*018]:

“…” “Double quote marks around the note.”

Type [Note \*018]: Parameter [Commentary].

## Now (Level) ---(----- Concept \*513 September 11, 2002

The ‘Now’ level is a scalar attribute level that is being tracked as a system evolves. It is a present moment report. It is a snapshot, or instant picture of the scalar level at a defined point in time, space and event condition. It is conceptually the same as a Past level. The difference is that we are indicating that the level of a real system is still probably at that level as far as we know.

Every time you check how well the product is doing during the development cycle you get a NOW level.

Related Concepts [Now \*513]

Past: \*106

Benchmark: \*007

Goal Step, \*620 ---)---

Type: scalar level, Benchmark

Keyed Icon (Now, \*513) ----(----

Note consistent with other benchmarks pointing to past , left ways like < and <<

Synonym:

• Present level \*513

Usage:

• “ the Now level of our product is 33 minutes.”

• It is a way of reporting progress during system evolution

• it is a way of announcing the present status of a system attribute level

## Numeric Tag "Numeric Tag" : Concept \*110. 10 January 2003

A term beginning with the ‘*asterisk ‘\*’* sign, and immediately followed by a number. The Numeric Tag is a human-language-neutral unique tag for each Planguage Glossary-defined term.

*Rationale: This \*Tag is intended to help identify Planguage concepts "Planguage concepts" unambiguously regardless of local natural language, local business type terminology, company terminology, dialect, spelling, translations, changes in terminology, or any other reason for variation. It is applied to all glossary defined terms and to related symbols, icons and Planguage constructs.*

*Example: This concept is number 110, so its Numeric Tag is \*110.*

Synonym: Glossary Concept Numeric Tag

## Object Concept \*099 January 8, 2004

An object is any system component that can be described in terms of its attributes; functions, *performance, and cost*; as well as past, present or future attribute states.

An object can be real or conceptual. It is a term we use for systematic building blocks of larger systems – which themselves might be viewed as more-complex objects. Any object can be composed of sub-objects. It is a matter of taste, and relative scope whether we use the term object or system to describe something.

Notes:

1. The object concept is extremely general. ‘Object’ is a description of almost anything.

2. The main difference between our definition, and common use of the term in the software field, is that we explicitly include the notion that an object can be described in terms of its *performance and cost attributes*. That, central systems engineering idea, is usually totally overlooked in the software world.

3. There is an implication in the use of the term ‘object’ that it is intended to be reused in some way, for example, to be a component of more than one system.

4. ENTITY ->OBJECT ---> ARTIFACT (SEE HIERARCHY DICTIONARY)

Object is a tangible entity; artifact is a man-made object.

Synonyms [Object \*099]:

• System \*145

An object is simply a ‘little system’ – a building block of larger systems. Object implies a potential component of a larger system. The only distinction between the synonyms is that object infers something relatively small, compared to a ‘system’.

Given these two terms are mainly used completely distinctly – two concepts have been allocated.

Related Concepts [Object \*099]:

• Artifact \*227

• Entity \*645

Keyed Icon [Object \*099]: “Same icons as for System \*145.”

[ ----->O----> ]

or

----->(<*mission tag*>)---->

Example:

---->( Update Address)---->

In this context the mission tag, ‘Update Address’ can serve as the reference name for the object, the *performance and cost* attributes of the object being implied by the arrows.

Type [Object \*099]: Specification Concept.

## Objective [noun] Concept \*100 May 10, 2003

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

## Objective [adjective] Concept \*656

Independent of a given set of values. Not ‘Subjective’

For example:

Objective Quality:

Quantifiable quality levels independent of the value judgement (good or bad levels) of defined stakeholders.

“The mean time to repair an average fault is less than 24 hours.”

Related Concepts [\*656]:

• Subjective [Adjective] \*657

## Objectivity Concept \*388. August 3, 2001

Being independent of Subjective influences, being a reality outside individual minds. Being closer to some real truth, independent of observers person or culture.

Related Concepts:

• See extensive discussion under ‘Subjectivity’.

Rationale:

Central point of this: there are degrees of both Objectivity and Subjectivity in all analysis, measurement and estimation processes. We should try to understand their elements, nature and degree. Then we need to hold up these factors in the light of our objectives, or in light of purposes of the analytical process.

Practical application: Planguage contains dozens of parameters and devices to help us communicate and understand the degree of objectivity or subjectivity of any specification.

## Objective Function: Concept \*619 September 7, 2002

“An objective function maps future sequences of states of the world to the value they have for specific stakeholders, in utility terms. Objective functions have various inputs from which to derive an overall utility value. The inputs can be called the dimensions of the objective function.

The assumed aim is to maximise the utility we get.

Applying our objective function (which could well be no more than a judgment) helps us identify particular sets of states that look highly desirable to us and which we may choose to take as goals. Goals (also known as objectives, targets, "a vision", and so on) are a mental device for creating and communicating plans.”

Source: Dynamic Management for an uncertain world by Matthew Leitch, 23 July 2002, version 1.1, http://www.matthew- leitch.supanet.com/dynamic

## Observability "Observability" : Concept \*176. "\*176.Observability" March 10, 2002

A component of the system state vector is controllable if there exists a control signal that can bring the system from the current state to a desired state. It is

observable if it is possible to compute or estimate the state from a

finitely long observation of the system output. <-Steve Poppe

*Concepts from feedback control theory, controllability, and observability.*

*Note 1:*

*The generic term ‘observable’ is intended to cover any means of observing a system including testing, measuring, seeing, sensing, feeling, hearing etc.*

*Synonym: Observable*

## Off \*626 February 16, 2003

Predefined State. Same as False.

Related To:

• State \*174

## OK: Okay, \*669 July 30 2014

The OK point is the level which is barely no longer considered Failure. The OK Range starts just above the Fail level, at the OK Point; and continues until the Goal point level. The dictionary definition of *‘satisfactory, but not especially good’* is suitable.

This concept is designed so that it allows us to have a ‘positive’ point (Tolerable, OK, Goal, Stretch), rather than ‘negative points (Catastrophe, Fail). OK is the lower (worst) boundary of the OK, area, where the upper boundary is the Goal level.

**Synonyms** for OK \*669

Satisficed \*669 (from Satisficing) suggested by Erik Simmons, Intel, Aug 2014

Satisfied ?, maybe this is closer to Gaol (fully satisfied, as opposed to minimally; need to research this)

“Thanks – you could also consider addressing satisficing in relation to the OK level. Herbert Simon (Sciences of the Artificial and other books) uses the term quite well, and we’re using it successfully here in context of my Solutions Thinking class. Satisficing = at least minimally meeting all

requirements. Useful in complex systems where even if an optimal solution exists and you happened upon it, you would not necessarily recognize its optimality, nor would you be able to prove that property of it.” Erik Simmons, Intel, August 12, 2014 Email to TsG

\*669 OK is **Related To**:

Fail \*089

Scale: \*132

Goal \*109

O---------|Tolerable------------------|OK---------|Goal----------|Stretch----- >

Dead Alive but Failure Good Success Incredible

O………Catastrophe|Tol.------------Fail|OK---------|Goal----------|Stretch---🡪

Dead |Alive | Good | Success | Incredible

Keyed Icon \*669 OK

Experiment 11 8 2014

O……….!!!!!!!!!!!!!!!OKOKOKOKOK++++++++&&&&&&

Dead Tolerable OK Success Stretch

† ☹ ☺ $ &

## On \*625 February 16, 2003

Predefined State. Same as True.

Related To:

• State \*174

## Open-Ended Concept \*101 February 26, 2003

‘Open-ended’ means that a system’s architecture: its interfaces and basic structure are of such nature that most system changes, no matter how unexpected, are relatively easy to handle without major effort.

The more open-ended, the easier the change will probably be.

Note: The degree of open-endedness can be measured by Adaptability metrics such as {Portability, Improvability, Extendability, Connectability}. (See also CE Section 5.6).

Type [Open-Ended \*101]: Adjective [System Architecture].

## Optimize Concept \*417. July 17, 2002

To optimize a design (‘design optimization’) is to reduce one or more resources, needed to achieve specified levels of performance, or stakeholder values. Or it is to improve the impact of designs in the direction of specified performance targets.

Note 1. The value-to-cost ratio is being increased when optimization, in the design or project evolution process, manages to identify designs, which reduce resources needed to achieve targeted system performance, or stakeholder values.

Note 2. Development process optimization would reduce the development resources {people, time, money} needed to meet product or system requirements.

Note 3. Product design optimization would reduce the resources needed to produce, operate and decommission (i.e. life cycle costs) the product or system, at defined levels of system-or-product values and qualities.

Type: Design process concept.

## Operational (adjective). Concept \*562. July 2, 2002

‘Operational’ refers to a state of defined systems and defined states of operation.

Usage:

Operational Requirements

Operational Attributes

Operational Functions

The operational requirements refers to the set of requirements which affect the defined system in defined states of operation.

Related Concepts [Operational \*562]

• Adaptive \*563 (adaptive attributes are things like portability, extendibility).

## Optimum Checking Rate Concept \*126 November 20, 2002

The average rate of checking a specification, which gives the highest productivity (effectiveness).

Note: The optimum checking rate is usually measured in terms of the ‘number of pages or lines per hour, which give the peak ability to identify major defects per page’, (peak effectiveness).

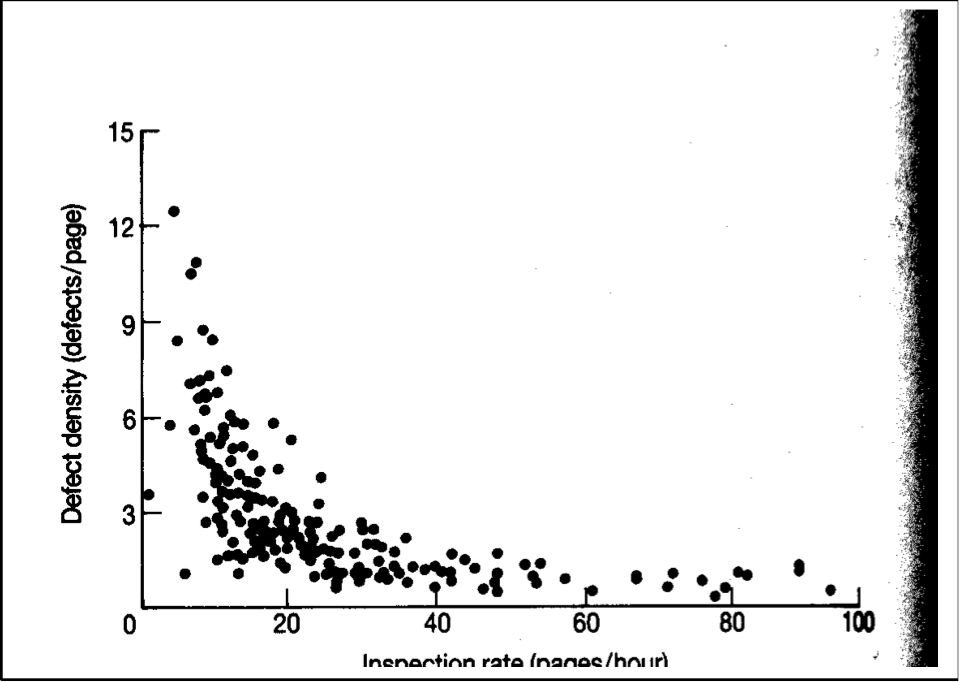


Figure (\*126): Graph showing how varying the checking rate affects the number of defects found [GILB93, Chapter 16].

[Note to Publisher: Bottom caption should say “Checking rate (pages/hour)”, <->delete this at publisher.]

Rationale: This measure is basic to making sure the SQC process is making effective use of people’s time. Without it, most organizations allow their staff to check too quickly, and find too few defects. They have the illusion of effectiveness but they lose out when the defects surface expensively downstream.

Related Concept:

•  Checking Rate (\*015): The rate checkers check a specification, whether optimum or not.

Type: Metric.

Domain: SQC.

## Option: Concept \*432. Dec 17, 2001 (spell corr)

The option is a short form of ‘template option’. It defines optional parameters or qualifiers in a model or a template for Planguage.

Note 1. Other uses for the concept are conceivable but not defined yet.

*Keyed Icon: parenthesis ( ) around the option.*

Scale: <the units of measure>.

(Meter: <the test process description or summary.> )

The Meter statement is optional, the Scale statement is not.

Type: Planguage specification concept [Templates]

## Or Concept \*514 February 5, 2003

‘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%.

Stretch [If Multinational | 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.

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

Example:

Goal [ {Europe | USA} & {Trials | Prototypes} ] 60%.

Assumption: (Deployment | Decommissioning), (Profit | Break Even).

Keyed Icon [Or \*514]: ‘|’ “As in A|B.”

Type [Or \*514]: Logical Operator.

## Or Better Concept \*550 March 5, 2003

‘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.

Examples:

Goal [Mechanical]: 60 degrees Or Better.

Planned Level [Electronics]: 60% -> +.

Wish: 60 -> +.

Stretch: 99.90% Or Better.

Survival [Offices]: 35 degrees C Or Better.

Related Concepts [Or Better \*550]:

• Until \*551

• Or Worse \*549

• Or Better \*550

• Range \*552

Keyed Icon [Or Better \*550]: -> + (a range towards a positive experience number)

Type [Or Better \*550]: Expression [Scalar Range].

## Or Worse Concept \*549 March 5, 2003

‘Or Worse’ is an expression used within a scale 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.

Examples:

Must Avoid [EU, Next Generation Product]: 50% Or Worse.

Fail [Banking Market]: 20% -> -.

Fail Level: 20 seconds -> -.

-!- [Households]: 20 degrees Centigrade -> -.

Fail, Fail Level and Must Avoid are synonyms \*098

Related Concepts [Or Worse \*549]:

• Until \*551

• Or Better \*550

• Range \*552

Keyed Icon [Or Worse \*549]: -> -

Type [Or Worse \*549]: Expression [Scalar Range].

## Output Concept \*636 June 27, 2003

An output (or system output) is the result of a system.

Notes:

1. Output has two primary uses in Planguage: it is either used to describe a function output (a performance attribute), or to describe a process output (a resource asset).

2. For intellectual processes, process outputs are always data. For physical processes (such as factory production), this can be extended to include material outputs or products.

Synonyms [Output \*636]:

• System Output \*636

• Product \*179

• Result \*130

Related Concepts [Output \*636]:

• Process Output \*179

• Performance \*434

• Input \*178

Figure \*179a: Outputs from a process are data and/or materials

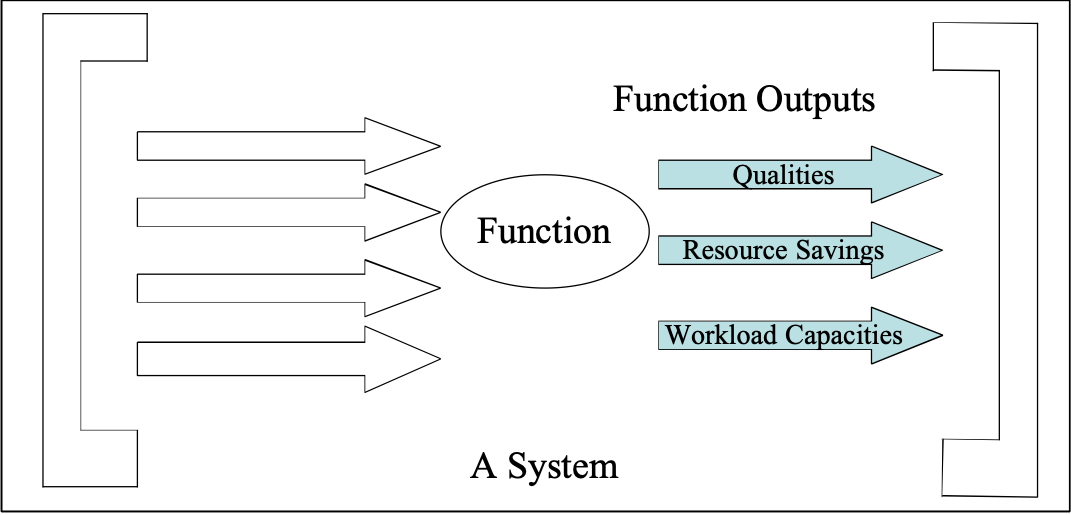


Figure \*179b: Function outputs, also known as performance attributes.

Keyed Icon [Output \*636]: O🡪 or ^[<*process name*>] 🡪 <*output name*>

“The icon is an arrow exiting a function oval, or an arrow exiting out of a process rectangle.”

Type [Output \*636]: Performance Attribute or Resource Asset (Asset).

## Owner Concept \*102 February 5, 2003

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.

For example: a system owner, a specification owner, a standard owner, or a process owner.

Notes:

1. An owner is responsible for updating or changing an object, including maintaining its control information (for example, Status, Version, Quality Level, and Location).

2. An owner will ensure the object adheres to any relevant standards.

Related Concepts [Owner \*102]:

• Stakeholder \*233

Type [Owner \*102]: Parameter, Role.

## Parallel Development Concept \*363 February 25, 2003

Any development of more than one Evo step, that takes place at the same time.

Notes:

1. Within Evo, given the need to keep up the frequency of deliveries, and the length of time it can take to develop and produce certain steps, ‘parallel development’ is a potential solution.
2. While it means *potential* loss of the advantage of being able to use feedback experience-data, it does mean an advantage in utilization of elapsed time.
3. With intelligent step planning the loss of useful feedback can be minimized.

“‘Partly simultaneous’ development of closely related product variants.

The purpose is to reduce the TBSP (Time Between Successive Products).”

[JANDOUREK96]

Type [Parallel Development \*363]: Process.

## Parallelity: Concept \*104. August 3, 2001

Having a number of parallel Evolutionary cycles.

These can be any type of Evo cycles frontroom or backroom (concurrent engineering).

Related Concept:

• Parallel Development, \*363

Domain: Project Management.Evo.Step Planning.Parallelity

## Parameter Concept \*105 March 4, 2003 B J

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. A parameter is defined as a part of Planguage.

Examples:

• Past Parameter [Scalar, Benchmark]

• Goal Parameter [Scalar, Target]

• Scale Parameter [Scale of Measure]

• Defined Parameter [Description]

• Risk Parameter [Risk Analysis]

• Impacts Parameter [Relationship]

• Authority Parameter [Specification Control]

• Note Parameter [Commentary]

• Version Parameter [Specification Control]

• Function Parameter [Attribute]

• Function Requirement Parameter [Requirement]

2. The master definition of most of the Planguage parameters is found in this Glossary.

See also [GILBWWW (www.gilb.com)] for additional, updated and new parameters.

3. 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.

4. Parameters are not user-defined terms. User-defined terms are defined by a project or organization, to describe the target system, organization or project (They are not part of the definition of a specification language).

Synonyms [Parameter \*105]:

• Specification Language Parameter \*105

Related Concepts [Parameter \*105]:

• Term \*151

• Project Language \*247

• User-Defined Term \*530

Type [Parameter \*105]: Specification Concept.

## Parameter Specification. Concept \*325. August 3, 2001.

A parameter specification is either the ‘Parameter Definition’, or the ‘Parameter Value’. It generally comes after the parameter name, after any [qualifier], and before any source information.

In the examples:

Example Tag:

Scale: *Up-time of [defined components] in [defined places] at [defined times].*

Meter [Acceptance Test] *Contract Condition 6.5* 🡨 Contract with ABC Dec 24th.

Goal [Software, GB, 2001] *60%* 🡨 Board policy

The parameter specifications, in the example above, are in italics.

Related Concepts:

• Parameter Definition \*323

• Parameter Value \*324

## Parameter Definition. Concept \*323. August 3, 2001.

The parameter definition is a set of words defining the parameter. It is a parameter specification.

Related Concepts:

• Parameter Specification \*325

• Parameter Value \*324

Scale: *Up-time of [defined components] in [defined places] at [defined times].*

Meter*[Acceptance Test] Contract Condition 6.5* 🡨 Contract with ABC Dec 24th.

Goal *[Software, GB,2001] 60%* 🡨 Board policy

The *italicized* expressions in the example above are parameter *definitions*.

## Parameter Value. Concept \*324. August 3, 2001

The parameter value is a numeric value, which varies along a defined scale, and which specifies the meaning of a given parameter.

Related Concepts:

• Parameter Specification \*325

• Parameter Definition \*323

*Scale: Up-time of [defined components] in [defined places] at [defined times].*

*Meter[Acceptance Test] Contract Condition 6.5* 🡨 *Contract with ABC Dec 24th.*

*Goal [Software, GB, 2001] 60%* 🡨 *Board policy*

*The ‘60%’ term in the example above is a parameter value of the Goal parameter. All parameter values are also classified as ‘parameter specifications’.*

## Parent Concept \*267 February 21, 2003

A parent is the *immediate superior* component to a kid.

Related Concepts [Parent \*267]:

• Kid \*266

Type [Parent \*267]: Role [Relationship].

## Participant "participant" : Concept \*238. August 3, 2001 "\*238.participant" .

A person or group who participates actively in a defined process.

## Past Concept \*106 March 5, 2003

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 \*106]:

• Past Level \*106

Related Concepts [Past \*106]:

• Benchmark \*007

Keyed Icon [Past \*106]: <

“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.

Type [Past \*106]: Parameter [Scalar], Benchmark.

## Pattern Concept \* 655 July 12 2006, April 12 2013 15:54

A ‘Pattern’ is a general tool that defines any idea structure, or sets of structures, in any intelligible format, which can be exploited by tailoring it to, or copying it into, a specific domain.

Patterns are a form of stored wisdom or knowledge. The main idea is to allow people to organize and reuse knowledge, rather than re-inventing it from scratch or memory.

Examples of patterns in Planguage and the CE book are: Principles, Scales of measure, Examples, Case Studies, Processes, Procedures, Concept Glossary Definitions, Entry and Exit Rules, Rules (for Specification), Standards.

Planguage as defined in Competitive Engineering is a set of diverse patterns intended to inform the user of useful structures which can either be copied or tailored to specific purposes and domains.

As defined by James Coplien;

“a piece of literature that describes design problem and a solution for the problem in a particular context‘

a three part rule, showing the relationship between a problem and a solution in a particular context **(J. Coplien)**

**Context** refers to a **recurring** set of situations in which the pattern applies.

**Problem** refers to a set of **forces** - goals and constraints - that occur in this context.

**Solution** refers to a canonical design form or design rule that someone can apply to **resolve** these forces.

**As defined by Wikipedia:**

<http://en.wikipedia.org/wiki/Pattern_language>

When a designer designs something—whether a house, computer program, or lamp—they must make many decisions about how to solve problems. A single problem is documented with its typical place (the [syntax](http://en.wikipedia.org/wiki/Syntax)), and use (the [grammar](http://en.wikipedia.org/wiki/Grammar)) with the most common and recognized good solution seen in the wild, like the examples seen in [dictionaries](http://en.wikipedia.org/wiki/Dictionary). Each such entry is a single [design pattern](http://en.wikipedia.org/wiki/Design_pattern). Each pattern has a name, a descriptive entry, and some cross-references, much like a dictionary entry. A documented pattern should explain why that solution is good in the pattern's contexts.

## Path Plan (Path): Concept \*534. March 30, 2002

A Path Plan is a series of two or more specified levels on a defined scale of measure from defined optional Benchmark levels and including any interesting series of defined constraint and Target levels. The path plan can be used to specify any interesting planned set of intermediary targets before a major target such as a Goal level is reached, and it can include that major target level.

Usage: it is particularly useful for describing our planned levels, and their benchmark/constraint context levels for a series of releases and for a set of evolutionary delivery steps.

Parameter: Path Plan or (abbreviation parameter) Path

Type: Target specifications.

Keyed Icon [Path Plan, \*534] <–v->

Example:

Path Plan [New Product, EU Market] {Past [Our Old Product, 2002] 60%, [Step = 1] 62%, [Step = 20] 70%, [Step= 40] 80%, Survival: 50%, Fail [UK, Schools] 70%, Goal [1st EU Wide Release] 85%, Goal [1 Year After 1st EU Wide Release] 95%, Stretch: 97%, Wish [Nordic Business] 99%}.

Path [Next Generation] {Past [New Product] 95%?, Goal [EU] 99%+}. <- Long Term Strategy

Type: scalar requirement specification

Related Concepts (Path Plan \*534)

• Step Level \*535 -v-

• Delivered level \*533 -•-

## PDSA Cycle: Concept \*168.

See Plan Do Study Act

## Percentage Impact Concept \*306 January 20, 2003

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 [Percentage Impact \*306]:

• Incremental Percentage Impact \*306

• Percentage Incremental Impact \*306

• %Impact \*306

Keyed Icon [Percentage Impact \*306]: %.🡪

Related Concepts [Percentage Impact \*306]:

• Incremental Scale Impact \*307

• Scale Impact \*403

Type [Percentage Impact \*306]: Scalar Metric.

## Percentage Uncertainty Concept \*383 November 18, 2002

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

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%

Keyed Icon: %.±? “Percentage & Uncertainty”

Type: Impact Estimation calculation

Domain: Impact Estimation.

## Perfect [Attribute Level] Concept \*648 August 12, 2004

A ‘perfect’ level of an attribute is a level that is flawless, complete and without defect of any kind. It cannot ever be improved under any circumstances. It is a benchmark concept.

Notes:

1. This is a benchmark concept. It can never be a valid target because we hypothesize that it can never be attained or proven attained (and guaranteed for future system use) in reality.
2. Perfect can be used as a benchmark parameter.
   1. Example:

Perfect [System Lifetime, All Conditions]: 100% Availability

1. The distinction between Ideal and Perfect is this. Perfect levels are always Ideal. But not all ideal levels are perfect. For example an Ideal market share might be less than 100% because of the threat of anti-monopoly action if it is larger. Similarly and ideal shareholding might be 51% to get control.
2. (Via Kai Gilb/Sri Sri Ravi Shakar) the concept of perfection depends on your chosen level of view. A system component that appears to have faults when viewed in isolation, may be part of a larger system which is able to operate at a faultless level, in spite of those component faults. The key idea ( developed only roughly here for now) is that the notion of perfect depends on viewpoint. It is not absolute.

Related Concepts [\*648]

• Ideal \*328

• Benchmark \*007

Type: Requirement Specification Benchmark

Note: Nov 14 2004> Could perfect be some notion related to the cost of attaining it. Perfect Effectiveness/Zero Cost <-Tom Gilb. Just started thinking this thought. Perfect Effectiveness (max effect) as distinct from Perfect Efficiency (Max effect/min cost). We need more work on this concept.

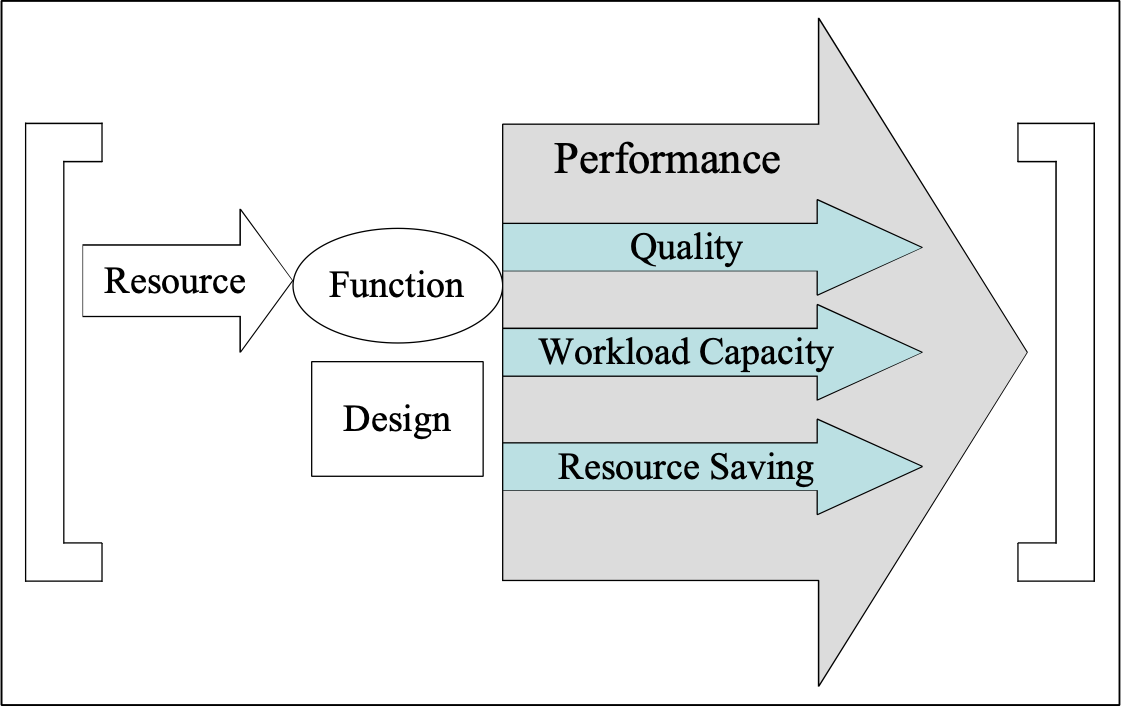
## Performance Concept \*434 June 5, 2003

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.

Performance attributes are scalar, and are of three types:

• Quality: ‘how well’

• Workload Capacity: ‘how much’

• Resource Saving: ‘how much saved’

Ill. [Performance]: performance characteristics can be classified into three major types. This is an arbitrary but useful distinction. Others are for example in the quote below.

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 1994

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.

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

• ‘work-capacity’ characteristics (such as storage capacity, maximum number of registered users, and transaction execution speed), and

•  resource saving characteristics (such as cost reduction, and reduced elapse times).

Related Concepts [Performance \*434]:

• Quality \*125

• Workload Capacity \*459

• Resource Saving \*429

• Performance Requirement \*453

• Performance Target (goal) \*439

• Performance Constraint \*438

• Benchmark \*007

• Target \*489

• Constraint \*218

Keyed Icon [Performance \*434]: O---> or O+

*Note:*

Keyed Icon [Resource \*199]: --->O or -O

Keyed Icon [System \*145 [Not Design]]: {Resource, Function, Performance,

Condition}:

[ --->O---> ] or [ –O+ ]

Type [Performance \*434]: Attribute [Scalar].

## Performance Constraint Concept \*438 February 27, 2003, 2019

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 sufficient to specify performance constraints.

In addition we have added ‘Tolerable’ \*539, and OK \*669 and others

Notes:

1. Stakeholders impose constraints. These stakeholders and their motivation should be explicitly documented together with the constraint specification (by 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 Constraint \*438]:

• Performance \*434

• Performance Requirement \*453

• Performance Target \*439

• Constraint \*218

• Fail \*098

• Survival \*440

Keyed Icon [Performance Constraint \*438]:

For Survival: O----------[-------------]----->

“One or both icons can be used depending on the numeric limits, which are being specified.”

For Fail: O ----!--------->

Type [Performance Constraint \*438]: Constraint [Scalar].

## Performance Design: Concept \*520. January 24, 2002

Performance Design is design specification aimed at, or as a side effect, satisfying performance level requirements. By definition this includes capacity requirements, quality requirements and savings requirements.

Note 1: On the multiple effects of any design.

Any design specification, however intended, in reality will impact many of our requirements. This multiple impact is true whether we like it or not. We cannot, it seems, only design on one pure requirement dimension (cost, performance, quality, condition constraint, function) without having some side effects in others.

Type: Design specification

Related Concepts [Performance Design \*520].

• performance \*434

* Function Design \*521
* Design process \*046
* (performance) target \*048
* (performance) constraint \*218

• cost design \*522

• function requirement \*074

• performance {architecture, solution, design idea, and other synonyms of design)

• design specification *\*586*

## Performance Survival limit: Concept \*477. August 31, 2001 tg 1546

A hard constraint on a scalar performance specified by the Survival limit parameter.

Related Concepts:

• Survival limit \*440

• Performance \*434

• Resource Survival limit \*476

Type: constraint, requirement

## Performance Requirement Concept \*100 May 10, 2003

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.

Figure \*100a: A performance requirement (objective) is specified as 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 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.

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

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 Penetrator = 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 [Performance Requirement \*100]:

• Objective \*100

Related Concepts [Performance Requirement \*100]:

• Requirement \*026

• Performance Target (goal) \*439

• Performance Constraint \*438

Keyed Icon [Performance Requirement \*100]: @

In context, -----@---->

Same as Target \*048

Note for the more-specific concept of performance targets use the set of target icons as follows:

O------>--->+--->?---->

A scalar performance arrow, ‘O------>’ with Goal (‘>’), Stretch (‘>+’) and Wish (‘>?’) icons

For performance constraints, use the set of constraint icons as follows:

O-------[------!--------]---------->

A scalar performance arrow, ‘O------->’ with Survival (‘[‘ and ‘]’) and Fail (‘!’) icons

Type [Performance Requirement \*100]: Requirement Attribute.

## Performance Target Concept \*439 February 27, 2003

A performance target 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%.

2. In contrast, a performance *requirement* includes both performance *targets* and performance *constraints*.

Related Concepts [Performance Target \*439]:

• Performance \*434

• Performance Requirement (Objective) \*453

• Performance Constraint \*438

• Target \*048

• Goal \*109

• Stretch \*404

• Wish \*244

Keyed Icon [Performance Target \*439]: @

“‘@’ is generic for any of the target icons, such as ‘>’ (Goal), ‘>+’ (Stretch) and ‘>?’ (Wish).”

In the context of a performance arrow icon: O------@-------->

Type [Performance Target \*439]: Performance Target.

## Performance to Cost Ratio Concept \*010 April 23, 2003

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 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 \*128 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 \*128. 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 [Performance to Cost Ratio \*010]:

• Sum of Performance \*008

• Sum of Costs \*128

• Value \*269

• Benefit \*009

Keyed Icon [Performance to Cost Ratio \*010]: +%/-%

Type [Performance to Cost Ratio \*010]: Metric.

## Philosophy "philosophy" : Concept \*217. August 3, 2001

"A basic theory concerning a particular subject"

Quotation:

"The set of values of an individual culture" <-The American Heritage Dictionary (Dell).

Planguage is based on the central idea that quantification of multiple critical quality values is basic to controlling real systems.

## Place Concept \*107 May 4, 2003 B

‘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.

Examples:

Place [Person Type]: {Buddist, 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 [Place \*107]:

• Space \*107

Related Concepts [Place \*107]:

• Time \*153

• Event \*062

• Qualifier \*124

• Scope \*419

Type [Place 107]: Parameter, Scope Concept.

## Plan Concept \*108 May 5, 2003

An idea, or more likely, set of ideas, created with a view to possible future implementation.

Notes:

1. A plan can contain requirements, design ideas and evolutionary project plans (Evo plans) for implementing Evo steps.

Related Concepts [Plan \*108]:

• Evo Plan \*109

• Plan [PDSA] \*169

Type [Plan \*108]: Noun, Management Concept.

## Plan (Parameter). Concept \*109. August 6, 2002

Synonyms: Planned Level (\*109), Goal [Parameter] (\*109), Planned Goal, Planned Budget

Keyed Icon [Plan [Parameter] \*109

---->------> a left arrowhead on a scale icon. Pointing towards the future...

Related Concepts [Plan, \*109]

• Budget Plan

## Plan [PDSA] Concept \*169 January 21. 2003

Plan [PDSA] is part of the ‘Plan Do Study Act’ Cycle (See Plan Do Study Act Cycle \*168). It involves specifying a theory or hypothesis, including a procedure for testing it.

The ‘Plan’ phase 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, it is the planning for the delivery of the step that has just been selected.

Related Concepts [Plan [PDSA] \*169]:

• PDSA \*168

• Do [PDSA] \*170

• Study [PDSA] \*171

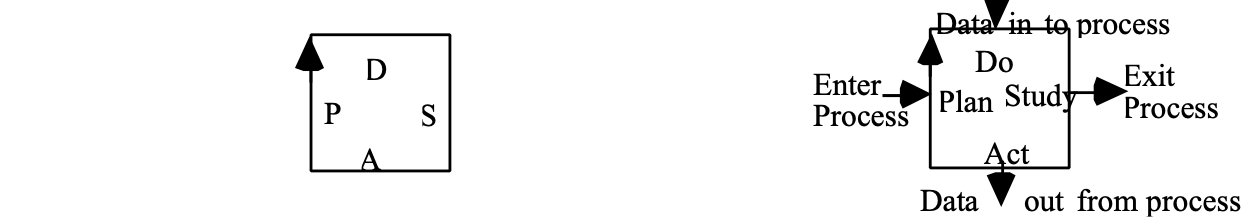
• Act [PDSA] \*278

• Evolutionary Project Management \*355

• Statistical Process Control (SPC) \*466

Keyed Icon [Plan [PDSA] \*169]: There is no keyed icon specification for Plan [PDSA].

Drawn Icon [Plan [PDSA] \*169]: Plan is the west side of the rectangular process symbol.



The four corners of the process icon stand for the Shewhart process-cycle definition of ‘Plan Do Study Act’.

Type [Plan [PDSA] \*169]: Process.

Domain [Plan [PDSA] \*169]: SPC.

## Plan Do Study Act Cycle Concept \*168 January 21 2003

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 [DELAVIGNE94, DEMING86, DEMING93].

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)

This is enabled by standardizing and stabilizing a target process (meaning; ‘wherever you carry out the improvemen’) so that the effects of any process changes can be credibly observed.

Deming

Acronyms [Plan Do Study Act Cycle \*168]:

• PDSA Cycle \*168

• PDSA \*168

Related Concepts [Plan Do Study Act Cycle PDSA Cycle \*168]:

• Plan [PDSA] \*169

• Do [PDSA] \*170

• Study [PDSA] \*171

• Act [PDSA] \*172

• Evolutionary Project Management \*355

• Statistical Process Control (SPC) \*466

Type [Plan Do Study Act Cycle \*168]: Method.

Domain [Plan Do Study Act Cycle \*168]: SPC & CPI.

## Planguage © Tom Gilb. Concept \*030 November 13, 2007

Planguage is a specification language and a set of related methods for systems engineering.

Planguage is defined formally in Gilb: Competitive Engineering [2005, Elsevier]. [www.Gilb.com](http://www.Gilb.com) contains samples and many related papers.

Notes:

1. Planguage specifically supports many key 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 aid charity project plans.

Planguage is designed to express ideas about 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 herewith when © is acknowledged.

6. If any reader finds the term ‘Planguage’ too ‘cute’, they may use the more directly descriptive ‘Planning Language.’ I often refer to ‘Planning Language’ before I introduce the term ‘Planguage’.

Synonym [Planguage \*030]:

• Planning Language \*030

Abbreviation [Planguage \*030]:

• PL \*030: Rarely used!

Type [Planguage \*030]: Specification Language & Method.

*Historical note: The term ‘Planguage’ occurred to me while watching a ballet with Michael Barishnikov and Susan Farrell at the Royal Opera House, Covent Garden in the 1980s <-TG.*

## Planguage Concept Concept \*188 2003

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

Planguage is defined formally in Gilb: Competitive Engineering [2005, Elsevier]. [www.Gilb.com](http://www.Gilb.com) contains samples and many related papers,

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, declared 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.

Examples:

Begrep [Norwegian Bokmål] = \*188 “Concept [Planguage, US].”

Tilstand [Norwegian Bokmål] = \*024 “Condition [Planguage, US].”

Marked [Norwegian Bokmål] = Market [HP 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]:

• Concept [Planguage] \*188

Related Concepts [Planguage Concept \*188]:

• User-Defined Term \*530

• Planguage \*030

Type [Planguage Concept \*188]: Concept.

## Planguage Term Concept \*211 March 12, 2003

A Planguage term is any concept name, which is defined as being part of Planguage.

Notes:

1. Planguage terms can be found in tailored variants or translations of Planguage (that is, in specific project specification languages).

2. Terms defined by projects to describe their user-defined terms are not Planguage terms. Planguage terms are valid across projects and organizations, which is not true of user-defined terms.

3. A Planguage term can have a numeric concept reference tag (such as this term ‘\*211’), but this is not mandatory.

Related Concepts [Planguage Term \*211]:

• Planguage \*030

• Planguage Concept \*188

• User-Defined Term \*530

Type [Planguage Term \*211]: Concept Name.

## Planned Level (parameter) Concept \*109. April 16, 2002

Identical option to Goal (Parameter) \*109. Goal is preferred use.

Abbreviation [\*109, Planned Level] Plan.

Note we also want to avoid using Plan as an abbreviation for Planned Level because of the word identity with concepts \*108 (a plan in general), **\***169 (Plan Do Study Act)

Rationale:

Plan level as a parameter has been in use in Planguage for decades. In 2002 we decided to replace it with the more-specific term Goal. This is due to the accumulation of many related scalar level concepts. Plan once stood alone as the general future level. But now we need to make a clearer distinction when using the specialized levels like Wish, Stretch, Fail, Survival, Maximum, Minimum etc. In the long term I would like to retire Plan entirely as a synonym to Goal. But for the present we can let it live on for those who prefer it. <-Tom Gilb in collaboration with Kai Gilb.

Related Concepts [Plan (Parameter) \*109]:

• Performance Target \*439

Synonyms: [\*109, Planned Level] : Goal [\*109]

Note Goal is our preferred parameter name. But as we have used ‘Plan’ in Planguage for decades, many people are accustomed to Plan and prefer to use it.

Keyed Icon [\*109] ----->-----> (a single right-pointing arrow on a Scale icon)

## Planning: Concept \*382. August 4, 2001

Planning is a human activity of ‘visualizing the future’, with the intents of

• being prepared for it and

• modifying future States to suit Stakeholder purposes.

Scope: Planning includes the activities of analysis, requirement specification, design, design analysis, quality control, and project management (with feedback to the planning process).

## Planning [SQC] Concept \*284 November 20, 2002

Planning is an SQC sub-process. An SQC team leader makes a ‘master plan’ for the specification quality control of a specific ‘main specification’. This planning process is usually of about an hour’s duration, and involves filling out a ‘Master Plan’ form.

The team leader’s decisions include:

• who will search for which type of defects

• which document and specification areas are to be checked

• which checking rate(s) will be used.

Note: This SQC sub-process should not be confused with other planning processes!

Source: A detailed explanation of the SQC Planning process is found in the Software Inspection book [GILB93].

Related Concepts: Planning \*382

Type: Process.

Domain: SQC.

## Planware: Concept \*577 July 12, 2002, Aug19 2010 added \*137

Planware is any specification, intent or agreement, which documents or reveals human intentions, and perhaps the background and status for those intentions. In simpler words, any plans of any kind. It does not have to be written.

Related Concepts [Planware, \*577]

• Dataware, \*576 any written planware can be computer held and is dataware.

• Infoware, \*578 Infoware is information of any kind, a small subset of which might be planware

• Documentation, \*579 planware is probably largely documented, but some plans are in peoples minds or agreed and presented orally, or made by default (no specific planning action)

• Logicware \*575 is the ‘active’ logic directing computers in their actions. To some degree plans and intents can be described in logicware – which in that case is a form of planware.

• Infoware \*578

• Specification Concept \*137 must be written tg aug 2010

## Platform Development: Concept \*362. August 4, 2001

A product development process based on products derived from a common cohesive, underlying architecture, and related synchronized development processes and organizational values [JANDOUREK96 (HP)].

Platform Development is an attempt to reuse 10% to 90% of the common elements when creating product variants. The ‘Platform’ refers to the base architecture and related common development processes for a set of related products.

## Platform Management: Concept \*363. August 4, 2001

Platform Management is a way to get a high degree of leverage, in multiple parallel projects, so as to improve the probability of positive net benefit, by maximum reuse of ‘architecture and development process’.

## Policy Concept \*111 February 22, 2003tg

A policy is a set of principles for decision-making, which permit delegation of decision-making to other people, at other times, under ‘unknown conditions at the time of writing the policy’. However, policy may be ignored for higher priority considerations. For example, because of a law or contract in conflict with the policy.

Example:

"All performance ideas shall be expressed in quantified terms".

Example:

Best:

Type: Policy.

Defined As: Our product quality and workload capacity will be ‘obviously’ higher in the 5 most-highly-rated user-quality areas. <- Corporate Quality Policy.

Authority: CEO.

Type [Policy \*111]: Standard, Parameter [Standard].

Domain [Policy \*111]: Standard Definition.

## Potential Consumer: Concept \*354 . August 4, 2001

An organization, or individual, which does not presently use or pay for a defined system {product, Evo step, service, process} but who might consider doing so.

*Rationale: The term ‘consumer’ was chosen rather than ‘client’ or ‘customer’ to avoid undue emphasis on the matter of direct payment. A potential consumer is one who might use something, irrespective or whether they directly pay for it or not.*

## Pragmatic (noun) "pragmatic" . Concept \*230 August 4, 2001 "\*230.pragmatic"

A ‘pragmatic’ is a device for getting something done, which is more *practical* than theoretical.

*We do not have to understand why or how it works, only know that it probably works. This would not satisfy a scientist, but is useful in engineering.*

*Heuristics are pragmatics.*

## Precedes: Concept \*445 July 24, 2001

‘Precedes’ gives information about required sequential *evaluation* of Planguage statements, terms or expressions, in relation to one another. It is implied that the result of the evaluation of any preceding statement or expression, must be applied to the following Planguage component evaluations.

Usage:

A Precedes B. or A: Precedes: B

Iconic ‘Use A (B)’ means B Precedes A in evaluation as a Planguage statement, and the results of B should be applied to the understanding of A.

Means; Statement A must be evaluated before B can be (is allowed to be) evaluated.

Related Concepts: After (time sequencing of Evo steps), Before time sequencing of Evo steps), Priority (tradeoff power of specifications)

“Prededes’ applies to any Planguage specification, and its *evaluation* sequence. Before/After are specifically limited to the *execution* sequence of Evo steps.

*Keyed icons: Use parenthesis ‘(...)’as in mathematics. To indicate terms which depend on one another (inside a parenthesis) and to indicate sequence of evaluation, both left to right time sequence, and innermost parenthesis can be done independently.*

*So for example*

Spec23: Spec Y(A, F, K) Spec X:(B, G, P, Spec Z (R,W)).

This indicates that specification Spec23 is composed of 8 defined sub-specifications {A…W}. Spec Y must be completed before Spec X, Spec Z can be done as a unit (R & W). Spec Y is composed of sub-specs A F and K which need to be completed before the Spec Y is considered complete.

In addition It can be used for sequencing steps:

Evo Plan 2005: (Step A, Step B), (Step C, (Step X, Step Y)) (Step GG).

This means that the Evo plan is that Step A and Step B must be done before the next set of steps can be done, and that they can be done in parallel. Step X and Step Y can be done in parallel. They can be completed before or after or in parallel with Step C. When all 3 (C Y Z) are done we can go on to Step GG. Separators such as comma or left arrow are permitted but superfluous.

Note: the parameters After and Before are explicitly designed to specify Evo step sequencing. ‘Prededes’ is more general, it applies to *any* specification, but that includes Evo step specification.

Keyed icon [Precedes] : (….) parenthesis to indicate sequencing.

The ‘->’ (impacts) could not validly be used. A->B means if you do A then it has an effect on B.

B(A) means A must be evaluated before B.

B ((A) C)) means evaluate A, then C, then B.

Type: Planguage parameter.

## Precision. Concept \*401. August 4, 2001.

Precision is a general specification quality. It is the degree to which a specification points out exactly and accurately the system components (e.g. stakeholders, design functions, sub-systems) for which the specification is *intended* by the Author to apply. And consequently, it is also the degree to which a specification *avoids unnecessary generalization*, beyond the system components that it is *intended* to apply to, by the author of the statement (or by the objective intent of the system).

Note 1. This is, at least, ‘subjective precision’.

One problem occurs when the author of a specification is precise with what they intend to say, but is incorrect with regard to what they should have specified objectively, in order to serve higher priority specifications (such as architecture or company policy or a contract). The specification is subjectively precise from the Author point of view, but objectively imprecise (from some Stakeholder’s point of view, or from the total system point of view). Objective imprecision is a form of inconsistency, and can be discovered by Specification Quality Control (SQC) when comparison of the specification is made with the higher priority specification (known as Source specification).

Specification Precision: Scale: The degree, in percentage of a defined scope of specification statements, to which a defined Type of specification is explicitly understood by a defined specification Readership, to apply precisely to the spec author intended System Components, and no others.

A possible scale of measure, or a general template, for specification precision.

*Requirement Precision* means that a requirement applies to specifically targeted stakeholders and system components only.

One of the most important tools for improving Precision in specification is the Qualifier.

*For example:*

*Goal [Stakeholders {A, B, C}, Components {D, E, F}, After {1st Release, or January Next Year}] 99.6%* 🡸 *European Law 52.3*

*Or*

*Design X [{Navigation [Emergency], Ground Control [Navigation]] Use Mil Spec GPS.*

## Price. Concept \*033. August 4, 2001

Synonym for cost.

## Price of Nonconformance Concept \*034 May 25, 2003 B

The price of nonconformance (PONC) is “What it costs to do things wrong” [CROSBY85].

PONC is the ‘cost to find and fix defects’ between two defined points in a systems engineering development process (See figure below for a list of what costs could be included).

Price of Nonconformance (What it costs to do things wrong)

• Reprocessing

• Expediting

• Warranty

• Unplanned service

• Returns

• Overdue accounts receivable

• Rework

• Downtime

• Scrap

• Computer reruns

• Explanation time

• Excess inventory

Figure \*034: Price of Nonconformance according to Crosby [CROSBY85]

Notes:

1. In general, PONC increases the later in the implementation process that a defect is found. The relative costs can be expected to be 20:1 at the test stage (20) compared to the development stage (1); and 80:1 at a field-use stage (80) compared to development stage (1) [GILB93].

Related Concepts [Price of Nonconformance\*034]:

• Process Improvement \*114

• Rework \*390

Type [Price of Nonconformance \*034]: Metric.

## Pre-requisite steps: Concept \*317. August 4, 2001

Evolutionary steps that are defined as pre-requisite by some Planguage mechanism.

*Examples of mechanisms are use of parameters Starts, Before, During, Ends, After*

*Example:*

*Step99: {Function-X, Design-Y[After Design-Z Ends], Design-X }.*

*Step101: {Design-A [Before Step99 Starts], Function-3}.*

## Principle Concept \*208 May 28, 2003

A principle is a short basic statement, which summarizes and teaches basic philosophy or the pragmatics of a method.

Notes:

1. An excellent collection of systems engineering heuristics are found in ‘The Art of Systems Architecting’ by Mark Maier and Eberhardt Rechtin [MAIER02].

2. An excellent discussion of the importance of the heuristic inthe engineering process will be found in KOEN03.

Related Concepts [Principle \*208]:

• Heuristics

Type [Principle \*208]: Guidance.

## Priority Concept \*112 June 6, 2003

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 (examples Goal, Budget), constraint levels (examples Fail, 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.

7. Priority is not the same as 'time sequencing'. 'Priority' means 'priority for getting limited resources'. ‘A => B’ (=> icon for Before) means A will be sequenced before B. But ‘A > B’ means A will be allocated resources before B, and B might never get resources if they run out. Also, the mere fact that ‘A > B’ does not necessarily mean that A will actually be scheduled for delivery before B, if B finally gets some resources. There could be reasons to deliver B before A if it does get resources. However, if real time, or ‘time to market’ are the only resource we are considering, then time sequencing and priority seem to be equivalent concepts.

Related Concepts [Priority \*112]:

• Level \*337

• Risk \*309

• Gap \*359

• Dependency \*189

Keyed Icons [Priority \*112]:

• > has greater priority than (for example, A > B)

• < has less priority than (for example, A < B)

• = has equal priority to (for example, A = B)

where A and B represent tags of resource consuming options, such as requirements or Evo steps.

A space is recommended on either side of the icon, ‘A < B’, not ‘A<B’.

• ‘{> | < | =} [\*]’ ,priority operator and resource qualifier example ‘ A > [ Money] B’ means “A has priority over B for Money resource”

Explanation: Any priority icon (‘>’ or ‘<’ or ‘=’) can be followed by any qualifier indicating the set of resources for which this expression is valid.

Type [Priority \*112]: Parameter, Design Concept.

## Priority Level Concept \*471. August 22, 2001tg 1332

Priority Level refers to the specified attribute level class {Wish, Stretch, Fail, Goal, Survival}

The final priority relationship is determined by *other* factors such as qualifier and Authority.

Priority Level \*471 : is the parameter used to describe the attribute level’s relative importance {Limit (1), Fail (2), Goal (3), Stretch (4), Wish (5)}

Here are some requirement levels, viewed as keyed icons.

------[a----->>b----->c------]d-----> O -----[e------->>f----->g-----]h----->

MEANING

[a Survival: a lower limit on resource use imposed by a stakeholder SA (example corporate policy for quality)

>>b Fail: a minimum acceptable level of resource acceptable value to a stakeholder SB (example a not unprofitable level)

>c Goal: a satisfactory - successful - level of resource for stakeholder SC (example: a competitive or profitable level)

]d Survival: an upper limit on resource use imposed by a stakeholder SD ( example corporate policy on single investments)

-On the system output side of things (‘O->’):

[e Survival: a lower limit on a system performance attribute, imposed by a stakeholder SE (example Government Safety Law)

>>f Fail: a minimum quality level imposed by a stakeholder SF ( example corporate quality policy or marketing)

>g Goal: a planned satisfactory value level desired by stakeholder SG ( example a market segment, to be competitive)

]h Survival: an upper limit on a system performance value as imposed by stakeholder SH ( example Government determined Monopoly ..........regulation or corporate policy about exposing ultimate capability to market and competitors too early) .

## Priority Signals: Concept \*515. June 28, 2002, edit \*112 11nov07.

Priority signals are any specification attributes, local, in the environment or inherited, which can be used to determine the priority (claim on limited resources) of one specification over others.

Synonyms: priority information, priority specifications, Planguage priority information.

Related Concepts:

• priority \*112

• immediate attributes

• inherited attributes

Icon [Priority Signals, \*515]: >

Example A > B means A has priority over B

Examples:

Qualifiers, Parameters {Authority, Priority, Risk, Owner, Source, Stakeholder}.

## Problem Concept \*270 March 17, 2003 D

A question to be considered, solved or answered.

Illustration \*270: A design ‘problem’ with multiple requirement considerations. The implication is that we need to look for a set of solutions (design ideas) that satisfies all the requirements simultaneously.

Notes:

1. In systems engineering, we are primarily concerned with the task of satisfying multiple function, performance and resource requirements, under given conditions. We classify this as a ‘design problem’.

2. Problems may be viewed as being in problem domains other than systems engineering: for example, ‘research’, ‘management’, ‘marketing’, ‘political’, or ‘environmental’. However, Planguage offers a general method for supporting solving any problems. Planguage specification language helps capture *any* requirements. One or more requirements (including any targets, constraints, assumptions, dates and locations) may constitute a complete problem statement.

3. The problem with problems… is that even though we may solve a simple problem as outlined here, real problems have many more dimensions of performance and resource (cost) to consider simultaneously. A real problem is so complex that we cannot figure it out very well in a pure design process. We need to try out our solutions step by step: seeing how each step works, and using the feedback from it to decide on the next incremental step to take. This is the approach of Evolutionary Project Management (Evo).

Related Concept [Problem \*270]:

• Design Problem \*654

• Problem Definition \*598

Type [Problem \*270]: Design Concept.

## Problem Definition Concept \*598 March 17, 2003 B

A problem definition is a statement of a problem that implicitly needs one or more solution options.

Related Concepts [Problem Definition \*598]:

• Problem \*270

• Requirement \*026

• Design Problem \*654

Type [Problem Definition \*598]: Specification [Design].

## Procedure Concept \*115 February 22, 2003

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” <- [quoted in MINTZBERG94 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}.

Related Concepts [Procedure \*115]:

• Task \*149: 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).

• Process \*113

Type [Procedure \*115]: Description [Work Process].

Domain: CPI.

## Process Concept \*113 February 23, 2003tg

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.



A process with its three main component processes.

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.

ISO 9000, 2000

Synonyms [Process \*113]:

• Work Process \*113

Keyed Icon [Process \*113]:

1. ^[...].

‘^’ represents the repetitive process cycle, and reflects the ‘up arrow’ of the Drawn Icon.

[…] reflects the rectangle of the Drawn Icon (the Plan-Do-Study-Act sides).

Example:

^[Process-Name-Tag].

Drawn Icon [Process \*113]:

A rectangle with up arrow on left side (illustrated below).



Related Concepts [Process \*113]:

• Task \*149: A task is not necessarily done by executing a process (a defined procedure).

Repetition of task is key to the need for defining a process.

Type [Process \*113]: Standard.

Domain [Process \*113]: CPI.

## Process Description Concept \*177 February 21, 2003

A process description is a process definition. It consists of the following components: {entry conditions, procedure, exit conditions}. The process description must also reference any rules that apply.

Notes:

1. A process description is a standard. The fundamental way of improving a process is for the process owner to change the process description, to reflect better practice, and then get people to follow the changed description.

Type [Process Description \*177]: Standard.

Domain [Process Description \*177]: CPI.

## Process Improvement Concept \*114 November 13, 2002

The systematic activity of

• determining the root systemic causes of human work process problems, then

• changing defined and stable work processes, so as to

• eliminate, or reduce, the human tendency to commit errors;

• and to thus improve 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)

Type: Process.

Domain: Continuous Process Improvement.

## Process Improvement Effort "process improvement effort" : Concept \*116.August 4, 2001 1512

The Resources used to improve a work Process[[4]](#footnote-4).

Type: resource class

Domain: engineering.quality assurance.process improvement.resources

## Process Improvement Suggestion Concept \*088 November 21, 2002

A process improvement suggestion is an idea to change a process, in order to improve its performance and/or cost. Suggestions should be directed to the relevant process owner for evaluation (to see if they have sufficient value-to-cost impact).

Usage: Process improvement is often stimulated by everyday engineering work, which uses standards (procedures, rules, checklists and forms) in everyday situations. Process improvement suggestions are, for example, explicitly sought from checkers during an SQC process meeting.

Type: process improvement device

Domain: Continuous Process Improvement.

## Process Input Concept \*178 June 16, 2003

A process input is any data and/or materials input into a process.

Notes:

1. Process inputs include such things as components and specifications.

Keyed Icon [Process Input \*178]: ->

In context, <process input defined here > -> ^[<process defined here>]

“A ‘defined data type’ as the input, then the ‘->’ symbol and then the ‘process icon.’”

Process Input -> ^[Process] -> Process Output.

Example:

{Initial Design Specifications, Requirements} ➔ ^[Evo Process {Build, Try, Study, Adjust Specifications}] ➔ Field Results.

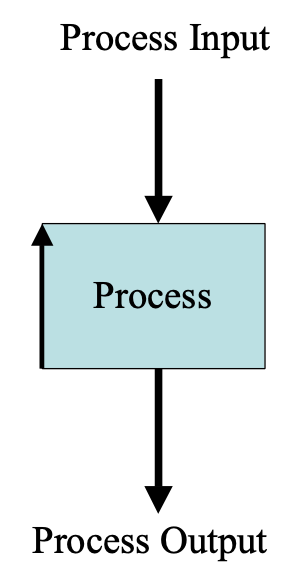
Drawn Icon [Process Input \*178]: Process input is represented by an arrow going into the top (or ‘north’) side of a rectangular process icon.

Figure \*178: Process Input

Related Concepts [Process Input \*178]:

• Process \*113

• Input \*331

Type [Process Input \*178]: Asset: Data or Materials.

## Process Language: (Concept \*246). August 4, 2001 1722

The subset of Planguage (or of a specific Project Language), which describes work activities. It is used to specify the process descriptions that utilize, or produce, the information expressed when using the Specification Language and Product Language.

*Usage: The Process Language is a standard that can be taught, reused and continuously improved by an organization or culture. It describes activity such as processes, procedures, rule evaluation, entry condition evaluation and, exit condition evaluation. These activities are to a degree supported and defined by glossary definitions and other defined terms. For example, ‘process’, ‘entry condition’ and ‘major defect’.*

Type: Planguage subset.

Domain: CE. Planguage.Process Language.

## Process Meeting Concept \*119 November 20, 2002

A process meeting is an SQC sub-process. It is part of the SQC Defect Prevention Process (DPP).

A process meeting is held following a specification meeting (after at least a short break) and is attended by the SQC participants. It lasts about 30 minutes and tries to capture grass roots insights into defect root causes, and to obtain practical process improvement suggestions.

About ten defects are selected. Each defect is analyzed for just three minutes (A minute to describe the defect, a minute to capture its possible causes and, a minute to suggest ways to prevent the defect). The meeting is conducted in a ‘brainstorming’ mode. Ideas are not criticized or finalized. That is left for the Process Change Management Team and their assistants to manage later.

Synonyms: {Causal Analysis Meeting [MAYS95], Process Brainstorming Meeting [GILB93]}.

Type: Process.

Domain: DPP.

## Process Output Concept \*179 June 27, 2003

A process output is any data or materials output from a process.

Figure \*179a: A process viewed as integral to a system. Function inputs are resource attributes, and function outputs are performance attributes.

Synonyms [Process Output \*179]:

• Product \*179

Related Concepts [Process Output \*179]:

• Process \*113

• Output \*636

• Input \*331

Drawn Icon [Process Output \*179]: A process output is represented by an arrow coming out from the bottom (south) of a process rectangle icon.

Figure \*179b: Drawn icon for a Process Output

Keyed Icon [Process Output \*179]: ^[<process tag>] -> <process output tag>.

“A defined process ^[Process X], followed by a right pointing arrow (->)

followed by a tag of the process output.”

Example:

{Initial Design Specifications, Requirements} ➔ ^[Evo Process {Build, Try, Study, Adjust Specifications}] ➔ Field Results.

Type [Process Output \*179]: Asset: Data or Materials.

## Process Output Metrics "process output\* metrics" : Concept \*121. August 4, 2001 1749tg

Measures of attributes of a specific process ‘output’ (see process output attributes).

*For example, the requirements coming out of a requirement specification process can be measured as to their Major defect content. They could contain 20 Majors per page, or one Major defect per page. A consistent quality or cost level of the output of a process is an indirect measure of the process itself, since it is due to the process.*

Type: System description concept, process description concept.

Domain: Process Language.Process.Output.Attributes,metrics

## Producer: Concept \*280. August 4, 2001

A ‘producer’ is the agent or process, which generates product from a process.

*One class of producer is ‘author’, who is a producer of written products as opposed to non-written products. Producers make products for customers and users.*

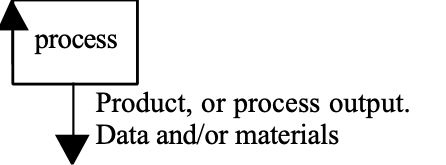
*Type: system agent class*

*Domain: system.process.agent.producer*

## Product "product" : Concept \*179. November 20, 2002

A product is an output of a process.

Related Concepts: By contrast the input to a process is called the process input (\*178)



The product ‘[Product]’ (of a function, ‘O’) needs to be distinguished from the attributes (‘->O->‘) of the product. For example, the actual ‘product’ of an engineering process can be a design specification. The qualities and cost associated with this product can vary; the design could have 20 Major defects, and cost three times as much as it should do.

Example: A ‘specification’ is the output, or product, of a specification process.

Synonym: Process Output, \*179, Output, \*179. work product, specification

Type: System component description.

Domain: System.Process.Output

## Productivity "product" : Concept \*659. August 6 2008 (New)

Productivity is **delivering promised value to stakeholders**.

**‘Deliver’** means actually measurable handed over and available to stakeholders.

**‘Promised’** means that clear written agreements, are made in contacts, requirements, documents and slides, or clear undeniable expectations are set.

**‘Value’** means something of perceived use, to the stakeholder; they need it, they want it, they are willing to sacrifice resources to get it, they will be unhappy if it is late or lower in power than their expectations.

**‘Benefits’** are the results of the perceived value to stakeholders. Benefits are what *really* happens, though time, as a result of the engineering value delivered.

It is an open question whether systems engineering should attempt to take some planning responsibility for enhancing benefits realization, or whether this is the system recipient stakeholders that should be responsible for planning an environment to maximize benefits. Someone has to take this responsibility, and I fear that the system users with their ‘day jobs’, do not feel they are responsible or capable. In which case an opportunity for systems engineers, to enlarge their conventional scope of planning, exists.

So, we can simplify and say ‘engineering productivity’ is the ability to deliver agreed requirements.

<- Gilb Paper, Engineering Productivity: Some ways to measure it and manage it. 2008 Unpublished. See www.Gilb.com

## Product Constraint. Concept \*474. August 22, 2001 tg 0024

Product Constraints are constraints that apply to the process output of an engineering process: the product or system being made by it.

Type: constraint

Related Concepts:

• Engineering Constraint \*473 a constraint on the engineering process itself, rather than the product.

Synonyms:

• System Constraint \*474

## Production Cycle Concept \*407 January 2, 2003

An optional backroom cycle, within the result cycle of Evo, concerned with product integration, or manufacturing and distribution of any products required for the delivery cycle.

Related Concepts [Production Cycle \*407]:

• Backroom \*342

• Result Cycle \*122

• Implementation Cycle \*123

• Development Cycle \*413

• Delivery Cycle \*049

Production is a synonym of Production Cycle \*407.

Type: Process.

## Profession: Concept \*207. August 4, 2001

An individual activity specialization requiring considerable training and/or experience.

*Examples, planner, engineer, quality control manager, project manager, top manager, supervisor, field worker of various types.*

*Type: Agent class and stakeholder class.*

*Domain: Organization.stakeholders.professionals*

## Progress Concept \*652 August 8, 2005 <|

Progress is scalar-requirement movement, from benchmarks towards desired and intended levels, like targets and constraints.

Related Concepts [\*652, Progress]

• Regress \*653

• Incremental Scale Impact \*307

• Scale impact \*403

• Range \*552

Keyed Icon [\*652, Progress] <|

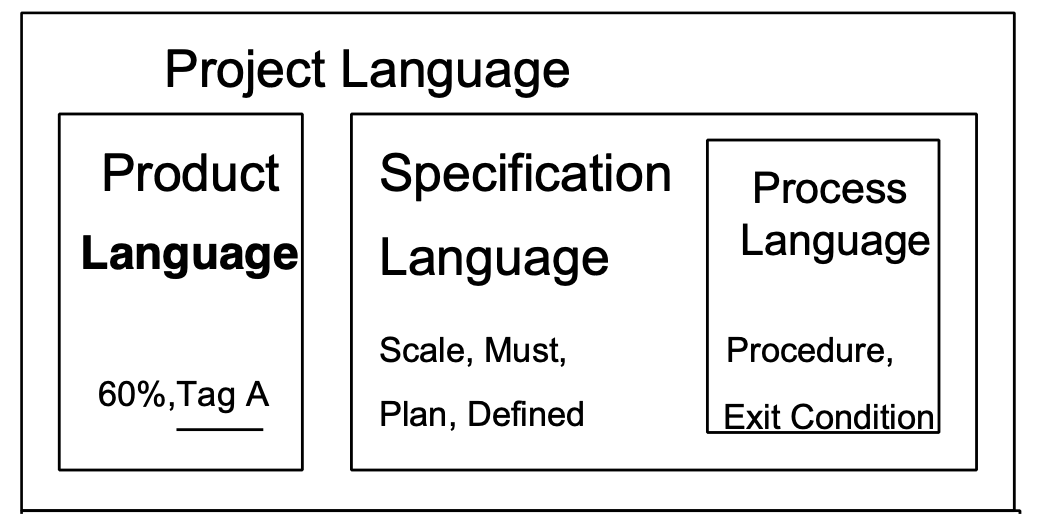
Example: O---<\_\_\_\_\_\_\_\_<| [Step 33, 66%] ----->---->

## Project Concept \*649 May 18 2005

A project is an activity to produce change to a defined system.

## Project Language: Concept \*247. March 4, 2003

The set of all Process Language, Product Language and Specification Language artifacts used by a defined project.



LB CHANGE PRODUCT LANGUAGE TO USER DEFINED TERMS

AND INSERT A TEXT ANNOTATION UNDER THE DIAGRAM

Note 1. The Project Language is a combined ‘team’ effort of the Specification Language combined with Product Language (‘our project’ specifications) and the defined processes in the Process Language (‘our best practices’ specification).

Type: Planguage set of languages

Domain: Planguage.languages

Related Concepts: [Project Language **\*247]:**

**• Specification Language \*248**

**• Process Language \*246**

• User-Defined Term \*530

• System Description Language \*183

Type [Project Language \*247]: ????

## Proposal \*587 July 16, 2002

A suggestion to be considered, evaluated, analyze, estimated before becoming accepted, approved or considered valid.

A proposal can be put forward orally, by specification, or be implied in some other formulation as a consequence (‘let us stick with the current design’, implies whatever that current design specifically is.)

Related Concepts [proposal, \*587]

• specification \*137 a written format for a proposal or something more accepted and approved than a proposal.

## Prototype. Concept \*187. May 27, 2004

A prototype is any design implementation that is made with one primary objective being to get feedback from stakeholders, including collection of data about costs and quality levels, for that technology/design, for those stakeholders.

A prototype reflects only conceptual design decisions and not decisions driven by implementation concerns.\*

Notes:

1. An Evolutionary delivery step is usually done with the intent of learning from the implementation. If the intent or practical situation is that we do learn from the implemented Evo step measure, and we do intend to change our plans if necessary as a result of the step feedback, then all such Evo steps are prototypes as we define prototype here.
2. the common implication of a prototype is that it is not industrial standard, and is primarily used to analyze potential design ideas. Evo steps are of an essentially different nature from other types of prototypes in that:
   1. we normally intend to keep the step in place if it succeeds.
   2. Some Evo steps are ‘scaffolding’ that we do intend eventually to replace, but they do allow us to progress other dependent Evo steps, while the replacement technology is being readied in the Backroom.
   3. The Evo steps are small enough ( about 2% of total budget) that if they do fail, we can afford to discard them, as with conventional prototypes. We can also afford a series of prototype steps until we get the results we want from a single step.
3. A ‘pilot’ implementation is a class of prototype, that is somehow of smaller scale of implementation that full implementation. The intent is to eventually scale up in volume or spread if and when it is successful. The intent is to avoid investment in full scale implementation if it is not successful. <- thanks Heidi Berg, ViaNova May 2004
4. Other primary objectives from an Evo Step ‘prototype’ in addition to getting stakeholder feedback, include delivering stakeholder value, and building delivery competence.

5. In Systems Engineering: Coping With Complexity, Richard Stevens et al define a prototype as "The experimental implementation of all or part of a developing system in order to gain information or assess risk." The authors provide a classification of the kinds of prototype:

1. imagination

2. simulation

3. emulation

4. partial implementation

Chapter 11 is on the topic of prototypes, and is quite good - among the best I've seen. <- Erik Simmons, Intel, Oregon May 2004

\*Paraphrasing of D. Harel, “Biting the silver bullet,” Computer (Jan., 1992) pp. 8-20 as quoted in Fredrick Brooks, Jr. The Mythical Man-Month Anniversary Edition, Addison Wesley 1995, p. 307 and 270 [Brooks95]

## Purpose \*528 February 23, 2002

The reason something exits, has been done, or is made.

Related concepts:

Rationale (parameter): \*259. a parameter for declaring information that justifies a specification.

## Qualifier Concept \*124 July 29, 2004

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.

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.

A qualifier defines any interesting set of *specific* time, location, and event conditions (also known as qualifier conditions). These are sometimes classified as ‘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!). *For example: Qualifier Tag: Qualifier [Condition1, Condition2]. This is useful when you want to define and reuse a set of qualifiers, especially a complex and lengthy set. For example: Goal [Qualifier Tag] 60%.*

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). We expect most Planguage parameter specifications to be 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.

Examples:

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 or 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 be all three 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.

Related Concepts [Qualifier \*124]:

• View \*484: Qualifiers can be used to specify a view of a system.

• Time \*153

• Place \*107

• Event \*062

• Condition \*024

• Scale Qualifier \*381: A scale qualifier is an individual condition qualifier within a scale definition.

• Scale Qualifier Definition \*446

• Indicator\*195

• Status \*174

Keyed Icon [Qualifier \*124]: [ ] “Square brackets around any set of qualifiers.”

Type [Qualifier \*124]: Condition, Parameter.

## Quality Concept \*125 July 26th 2006, 19June2013 (..as viewed by… stakeholder)

A quality is a scalar attribute reflecting ‘how well’ a system functions as viewed by a defined Stakeholder.

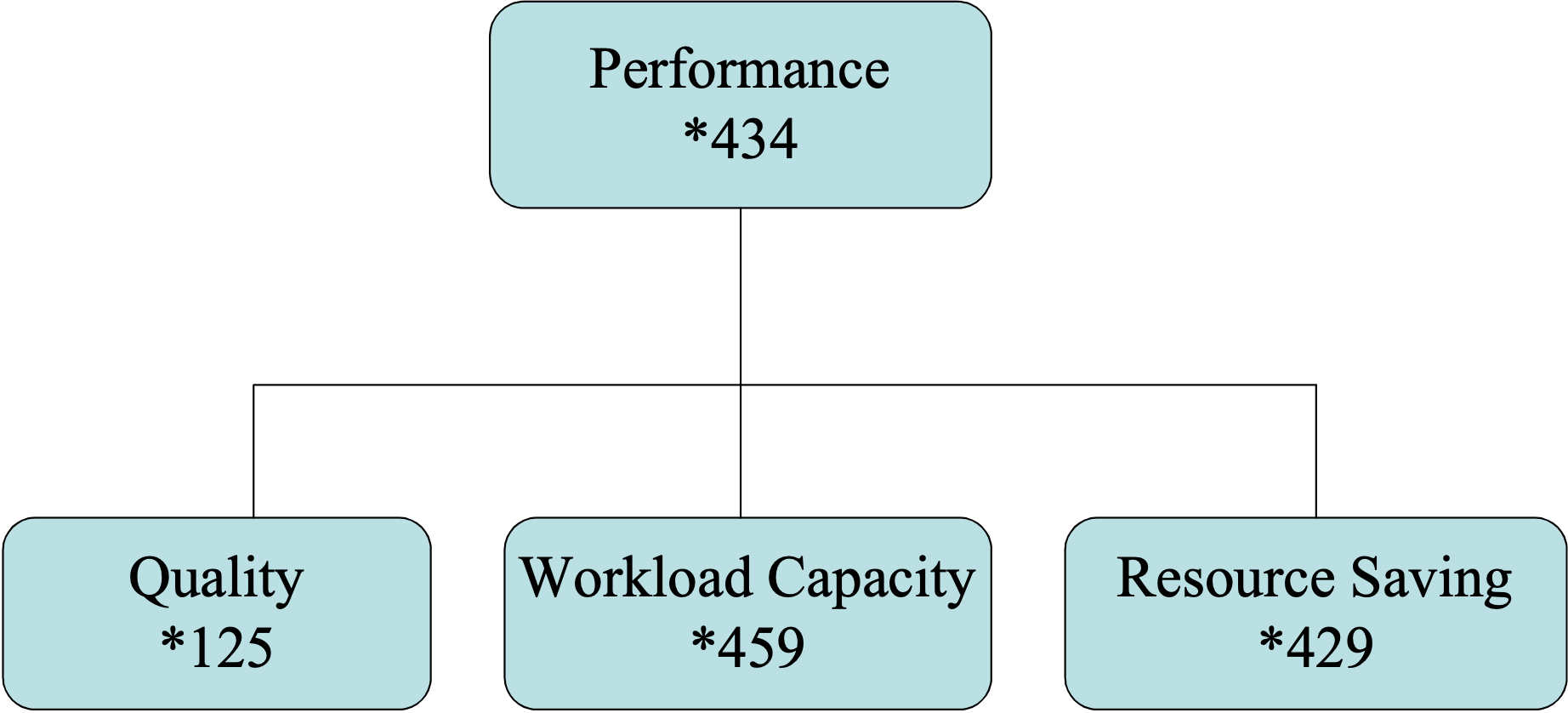


Figure \*125: Quality viewed in the context of Performance.

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.

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 (100%), 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 [Quality \*125]:

• Performance \*434

• Resource Saving \*429

• Workload Capacity \*459

• Quality Requirement \*453

• Value \*269

URL: here is a link to other opinions of defining the word quality:

<http://www.qualitydigest.com/html/qualitydef.html>

Types [Quality \*125]: Performance Type.

## Quality Assurance Concept \*282 February 21, 2003

Quality Assurance (QA) is the generic name for any set of activities, which have as their primary or partial intent, or effect, to influence (‘assure’) the quality levels of a product or process.

Notes:

1. QA includes, but is not limited to, quality control (testing, Specification Quality Control and more). QA, in particular, would include all concerns of this book, such as quality specification, design to reach quality levels, Impact Estimation of design impact on quality, evolutionary evaluation of quality levels, and SQC. QA even includes cost management aspects, since ‘cost of quality’ is an important aspect of quality.

Acronym [Quality Assurance \*282]:

• QA

Related Concepts [Quality Assurance \*282]:

• Specification Quality Control (SQC) \*051

• Impact Estimation (IE) \*283

Type [Quality Assurance \*282]: Process [Quality Assurance].

Domain [Quality Assurance \*282]: Quality Assurance.

## Quality Constraint "quality constraint" : Concept \*305. January 9, 2002

Quality constraints set limits on how bad, or how good, a quality level is allowed to be. On ‘how well’ the system should be.

Note: when analyzing the specific stakeholder situations, and the specific design costs, we might well encounter situations where the a quality constraint is violated and something needs to be adjusted. So, either an exception needs to be made, or the quality constraint needs to be adjusted accordingly.

Related Concepts:

There are several types of quality constraint:

• global quality constraint – applies within its specified qualifiers, for the system or project.

• fundamental quality constraints. Within its scope, it probably has priority over all specific types of quality constraint. It is at the base of a system development, before detail provides other constraints, some of which might have a higher priority – as determined by their qualifiers and parameters; such as ‘Authority’).

• performance constraints; expressed with a Limit specification, or a suitable constraint (like ‘Respect all valid national laws in country of export’)

• generic quality constraints like ‘quality levels must be high enough to get certification from electrical, electronic and Communications authorities in country of use, sale and export’. These are ‘generic’ because they generate specific constraints depending on countries of export and their current constraints on quality.

• failure-avoidance quality – usually expressed by one or more ‘Fail’ level specifications.

Weakly-related Concepts: These following quality targets are not generally classed as constraints, but rather as attractive targets – not brakes on the development. However, at the extreme, all requirements constrain the thinking of designers to some degree. So, we could theoretically add:

• success quality – expressed by Goal levels, and to some degree by Stretch and Wish levels.

But these are normally classed as quality goals, not constraints. They only constrain in the sense of saying ‘you do not need better design ideas than those which will meet these goals together with all other concurrent requirements’. The designer is probably not motivated to stretch further than these targets suggest. In fact that is the purpose of the Stretch Goal: to motivate designers to go some distance beyond the Goal levels.

*Examples:*

*Competitive Quality Policy:*

*Constraint: All product quality attributes must be defined at a value at least as high as we expect the competition to be, when our product is in a given market; and the quality levels should even then generally be noticeably higher.*

A global (‘all attributes’) and generic (‘at least as high…’, ‘higher’) quality constraint.

*Design Policy [Telecommunications]:*

*Constraint: No design idea may have a ‘worst case’ (‘estimate’ minus ‘the negative uncertainty %’, times ‘credibility factor’ (0.0 to 1.0)) risk level of quality that could threaten achievement of the Goal levels in <Critical Markets>.*

A global (‘No design idea’) quality constraint.

*Fail [European Market, Product Lifetime] 10% more than <Competitors’ Levels>.*

A specific (European…, Product…, 10% more) and generic (Competitor’s Levels) quality constraint.

## Quality Control (QC) Concept \*279 November 20, 2002

Quality Control (QC) is the generic name for a set of processes that try to check the quality of a work product compared to applicable ‘standards’ (like ‘rules’, which define acceptable quality.)

QC includes testing, Specification Quality Control (SQC), and any other activity such as meetings, hearings, customer feedback and surveys, which can result in discovery of defects in products. It does not include other forms of quality assurance (QA) such as ‘design’ and ‘process improvement.

Abbreviation: QC.

Related Concept: Specification Quality Control (SQC) \*051.

Type: Process.

Domain: Quality Assurance.

## Quality Requirement Concept \*453 March 20, 2003

A quality requirement is a type of performance requirement that deals with ‘how good’ a system needs to be.

Related Concepts [Quality Requirement \*453]:

• Quality \*125

• Requirement \*026

• Performance Requirement (Objective) \*100

Type [Quality Requirement \*453]: Performance Requirement Type.

## Quality Status[Specification] Concept \*463 November 28, 2002

The Quality Status parameter for a specification refers to the official determination of quality approval. Local concepts can be used. Generally the classes are {Not Exited, Exited}. Other possible categories are listed below.

Synonym: \*463 Quality Status

• Status \*463 (in the context of a specification or set of them)

Related Concepts: \*463

• Quality Level \*360

• Status \*174

• Maturity \*555

Usage: This can be used for a document, or any defined specification space, including a single requirement or design specification. Suggested settings include:

• Undetermined

• Under Revision

• Exited

• Approved

• ‘initial, defined, agreed upon, released’ (used by Damlier Chrysler 2002) for ‘working state

• Validated (passed Spec Concept Review)

• Verified (proven to be present and correct by some form or test or observation)

## Quality-to-Cost Ratio "quality-to-cost ratio" : Concept \*010: August 5, 2001 0044

A ratio of the design-impacted quality improvements, to the design costs to get them.

Rationale: Such ratios allow comparison of different design ideas.

Usage: In Planguage we differ from conventional usage of this term in that:

• we assume that both qualities and costs can be expressed numerically as requirements, and design ideas can be estimated for impacts in terms of multiple measures of qualities and of costs

• any useful set of one or more attributes can be chosen at any time to represent an interesting set of qualities or costs

• the costs are any idea of resources, and not limited to financial costs

Synonym: Quality/Cost ratio. Performance-to-cost ratio. Near synonym ‘Performance Efficiency’.

Related Concepts:

• Performance-to-Cost ratio can be used when any of the factors are Quantifiers or savings, and not just quality.

• Similarly valid: Quantifier-to-cost ratio, Savings-to-cost ratio.

• Popularly (USA) “Bang for the buck”.

Abbreviation: Q/C ratio.

Type: Design metric.

Domain: Engineering.design.analysis.metrics.Q/C ratio.

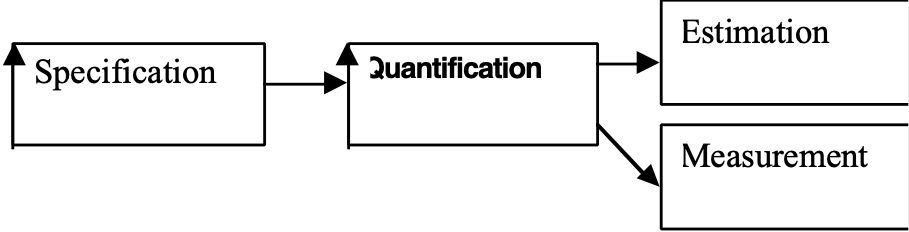
## Quantify, To Concept \*385 May 9, 2003 B

To quantify is to specify numerically.

Notes:

1. To articulate a variable attribute, using a defined scale of measure, and specifying one or more benchmark or target 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 pre-requisite for other processes, such as Estimation, Measurement, Testing, and Feedback Control.

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; measure2. To express in quantitative terms, or as a numerical equivalentLogic 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™ College Dictionary (Third Edition).

Our definition of quantify ((identical to variant 2 above) does not admit to the term ‘measure’ which is contained in this definition (variant 1), and in common use of the words. For us ‘measurement’ is a quite distinct activity (like variant 1 above), based on the quantification, (but not identical to it).

Related Concepts [Quantify, To \*385]:

• Scale \*132

• Meter \*093

• Measure, To \*386: To measure is to determine the numeric level of a scalar attribute, using a defined process.

• Estimate, To \*059

Type [Quantify, To \*385]: Verb.

## Quantity Concept \*437. January 3, 2003

A Quantity describes ‘How much’ a system performs or consumes.

A quantity is an attribute measured in terms of Capacity (\*459) or Resources (\*199).

A ‘quantity’ is type of attribute which we would not normally call a ‘quality’ (\*125)

Instances [Quantities]:

• Performance attributes: speed, work capacity, response time, weight, space, throughput • Resource attributes: time, money, data, space, effort.

Usage: ‘All system quantities can have quality attributes attached to them.

For example the quantity attribute ‘responsiveness’ can also be characterized by quality attributes such as reliability, variation, measurability, and testability.

Keyed Icon [Quantity, \*437]: ----> (the difference from other variable attributes like quality being in the Scale definition itself; using the ‘quantity’ definitions above).

Related Concepts [Quantity \*437]:

• Resources: are quantities. \*187 & \*199 used to specify cost budgets.

• Performance (\*434) goals are a class of quantity.

• Savings goals \*429 are a class of quantity

• Workload \*459 is a class of quantity.

• Quantity Goal, \*464 : is an umbrella class of performance goals which covers just about all other goals than those we class as Qualities.

Rationale: conventional thinking assumes that qualitative ideas cannot be quantified. We are trying to emphasize that there are at least two kinds of measurable attributes: qualities and quantities. Quantities being the conventional variables of performance and resources which we conventionally expect to quantify and to measure. See [CROSBY97] Alfred W. Crosby, The Measure of Reality. Quantification and Western Society 1250-1600, for a historical perspective of how society gradually absorbs more quantification.

Type [Quantity]: system attribute class, noun

Domain: system.attribute.quantity

## Quantity Goal, Concept \*464. February 23, 2002

is an umbrella class of *performance* goals which covers just about all other goals than those we class as Qualities. Any goal which expresses ‘how much’ rather than ‘how well’ is a quantity goal.

Type: Requirement

Note 1: Quantify goals include {Workload (\*459) Goals and Savings (\*429) Goals}

Domain: Engineering.Requirements.Performance.Quantity.Goal

Rationale: we needed a very general concept to distinguish Performance Quality goals from most others and the common denominator they had was that they were conventionally and easily quantified.

Related Concepts:

• Quantity Objectives: an objective is complex, and can have one or more goals, and one or more sub-objectives. A goal is an elementary requirement with a single scale of measure to define it.

## Question of Intent Concept \*301 November 20, 2002

A Question of Intent is a question concerning the technical intention of a Specification, directed to a Specification Writer. It is asked orally during the SQC Specification Meeting. It is immediately logged in the Editor Advice Log by the Scribe. It is answered orally by the Writer at the *end* of the Specification Meeting and so the question and answer are available to the editor during Edit.

Source: The ‘question of intent’ was invented at my client Douglas Aircraft Co. (now Boeing) in 1988. It was used as a diplomatic way for junior engineers to hint to senior engineers, that their engineering specifications were not totally understood, without suggesting that there was any engineering defect involved. The specification writer/editor may use the question of intent in the Editor Advice Log as a memory jogger. There is a need to consider answering the question, by either adding material to the written specification, or by rewriting part of it: the aim being to remove the need for anyone else in the future to have to ask the same question.

Type: process device, Editor Advice Log type classification

Domain: SQC.

## Range Concept \*552 August 8 2005

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

*The ‘doughnut’ diagram indicating different range concepts.*

Notes:

1. Range, as used in the figure above, 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, from Goal Level and better.

• Failure Range – some degree of problems or failure. Icon ‘!!!!!!!!!!!!!!’ (see icon <! For negative Incremental Progress \*307)

• Catastrophe Range – forget it! No use and no hope! Icon ‘……….’

• Tolerable Range: Acceptable Range – we can live with it, just above failure range, just worse than success range – i.e. just worse than Goal Level. Icon >>>>>>>

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 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 range 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 [Range \*552]:

• Landing Zone \*605

• Gap \*359

• Success Range \*548

• Acceptable Range \*545

• Failure Range \*546

• Catastrophe Range \*602

• Tolerable Range \*539 + \*532

• Tolerable Limit: Worst Acceptable Case \*539

Keyed Icon [Range \*552]:

Range uses several different icons as follows:

• For scalar arrow ranges outside the current targets and current benchmarks: ----

• For ‘Gap’ ranges: ==== (between defined benchmarks and defined targets)

• For ranges of achieved performance delivery or resource usage: •••••

• To express the current level on a range (impact estimate/the latest benchmark): |

see Incremental Scale Impact \*307 and Icons <| and <!

• To express an incremental change (\*307) we can use an underline (\_\_\_\_) between a benchmark and an incremental scale impact (\*307) level.

Thus: O-------<\_\_\_\_\_\_\_\_\_\_<| [Step 3]------>------> for a positive impact (<|)

And : O------[Step 4] <!\_\_\_\_\_\_\_\_\_\_\_<------------------->-------> for a negative scale (<!) impact

Example:

O----------<============>----> “A gap between a Past and Goal (===).”

O-------<•••••••••••|=======>-----> “Progress (•••) to current level (|) and a gap to Goal (===).”

Type [Range \*552]: Scalar Concept.

## Rate Concept \*139 November 20, 2002

‘Rate’ refers to the defined speed, with which a defined task, is carried out, per defined unit-of-volume worked on.

Example:

The optimum checking-rate for Quality Requirements, at Division ABC, by experienced engineers was 0.3 pages (of 300 non-background words) per hour, last year, on average.

This is merely a structural example not a fact!

Notes:

1. Rates can be set as process standards. This implies that adjustment of rates of work towards more effective or optimum, rate-levels is a thing that can be ‘learned’ statistically, by an organization, in order to improve their own work processes.

2. The most important rate, for effectiveness in identifying major defects, is the optimum checking rate. This is the best speed for checking a management or engineering specification, including the time needed to check it with respect to all prior related specification, and to all official standards for writing the specification.

This rate can usually be observed to be in the range 1±0.9 Logical Pages (of 300 non-background words) per hour of analysis. This is the rate that gives the greatest major defect-finding effectiveness.

Related Concepts:

• Checking Rate \*015

• Optimum Checking Rate \*126

Type: Metric.

Domain: Systems Engineering.

“I’ve always believed that when the rate of change inside an institution becomes slower than the rate of change outside, the end is in sight. The only question is when”.

Jack Welch, ‘Jack: Straight from the Gut’, [Welch01] p. 432

## Rationale Concept \*259 January 8, 2003

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.

Note: The information can concern the logic, the politics, the economics or whatever is of interest to declare in order to explain and justify another specification.

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?)

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 Rationalestatement. 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.

“Their’s is but to reason why: The value of recording rationale.”

[HOOKS01, Chapter 8]

Synonyms [Rationale \*259]:

• Justification \*259

• Reason \*259

Type: Parameter, risk analysis.

Historical Note: ‘Rationale’ is a term I found used by Synopsys, CA USA 1996.

## Readership Concept \*295 June 4, 2003

The readership (or intended readership) of a specification is all the ‘types of people’ we intend shall read or use the specification.

Note: 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 [Readership \*295]:

• Intended Readership \*295

Type [Readership \*295]: Parameter.

## Real Values: Concept \*350. September 29, 2002

Real values are estimates in impact estimation based on the ‘real’ defined Scale of measure.

*Related Concepts: Contrast with percentage-of-target values. The % values are a percentage of travel along that scale with regard to a specified baseline (0%) and a specified target value (100% concept).*

*Synonym:*

*• real impact*

*Type: Impact Estimation metric.*

*Domain: CE.Design.Impact Estimation.Real Value scale.*

## Recipient: Concept \*341. August 5, 2001 2240

An Evo delivery step recipient. A host system user, a customer, a client, a consumer, a stakeholder, a market or any people or instances that are the target, or potential target, of an Evo step delivery.

Related Concepts:

• Frontroom, \*343 (the project management processes where results are delivered to recipients).

Type: Planguage parameter, Evo stakeholder concept.

Domain: CE.Evo.stakeholders.recipient.

## *Record << Concept \*127 March 5, 2003*

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.

Example:

Usability:

Ambition: More user-friendly than <competitors’ products>.

Scale: Average Learning Time for 10% Slowest Learners in User Population.

Trend [Main Competitor, Next Year]: 1 minute.

Record [2001, 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 information for all scalar levels, see Source (<-) in the Record specification.

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.

Synonym [Record \*127]: Record level (see level \*337)

Related Concept [Record \*127]: Benchmark \*007.

Keyed Icon [Record \*127]: <<

Explanation: Arrow points towards the ‘past’ as in Past benchmark, ‘<’, but doubled for emphasis to show that this is an extreme benchmark. Usually used in the context of a scale arrow (---<<---).

Type: Parameter, Benchmark.

## Regress Concept \*653 August 8, 2005 <!

Regress refers to a scalar level that is worse than a defined benchmark.

Related Concepts [\*653, Regress]

• Progress \*652

• Incremental Scale Impact \*307

• Scale impact \*403

• Range \*552

Keyed Icon [\*653, Regress] <!

Example: O---[Step 34, -66%] <!\_\_\_\_\_\_\_\_< ----->---->

## Relationship Concept \*142 June 6, 2003

A relationship is a connection between objects.

Related Concepts [Relationship \*142]:

• 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

• Impacted By \*412

• Supported By \*414

• Supports \*415

Keyed Icon [Relationship \*142]: |-|

For example, A |-| B. “A is in a relationship to B. There is no notion of impact - just connectedness.”

Example:

Quality Q:

Scale: MTTR.

Goal [By End of Decade]: 30,000 hours.

Relationship: {Function F, Value Y}.

|-| Function G.

Type [Relationship \*142]: Specification Concept, Parameter.

## Relative Specification: Concept \*402. November 5, 2002

A ‘relative’ specification is a Planguage specification where the exact interpretation is dependent on the meaning of one or more other specifications.

Note: requirement specifications like ‘improved’, ‘better’, ‘world class’ Are ‘relative specification’ adjectives. They may be applied in a summary statement such as Gist or Ambition. But for testable clarity relative adjectives need to be finally defined using a Scale statement and a benchmark (like Past) and a target (like Goal).

Related Concepts:

• Absolute Specification \*509 Example ‘Year = 2009’.

Type: relationship concept.

## Remaining Major Defects/Page Concept \*060 March 11, 2003

This is an estimate of the major defect density remaining in a specification after SQC has evaluated its quality. This can be done before or after removing defects identified. This can be done based on a sample.

Notes:

1. The estimate is based on:

• defects found (using knowledge of % defect-finding effectiveness) and

• defect removal (using knowledge of % effectiveness of the correction process).

See Section 8.6 for a worked example.

2. The defect density is usually expressed as the number of remaining major defects / (logical) page.

3. It is important to understand that we do not know precisely *where* such defects are located, and we do not know exactly *what* they are. This is an estimate of the current total of the defect quantity remaining, before or after editing or debugging.

4. The estimate seems to be accurate to about plus or minus 30%. In rougher terms, experience shows that for every defect, which SQC finds and fixes, there is about one more left, which will be discovered by future QC (test, or another SQC) or customer use.

5. The ‘remaining defects’ estimate is primarily used to make an exit decision from a test or QC process. The ‘remaining defects estimate’ represents calculable future costs. One Major defect cured saves about 10 hours downstream with about 25%-35% probability. These downstream costs can be weighed against the cost of removing defects immediately, before proceeding further.

For every defect you find and fix there remain that many more in the system.

If that doesn’t work for you, try Gerald Weinberg’s Principle:

“There is always one more bug remaining”

(Even after you think you have removed the last bug).

Related Concepts [Remaining Major Defects/Page \*060]:

• Logical Page \*103

• Quality Level \*360

• Specification Defect \*043

• Major Defect \*091

Type [Remaining Major Defects/Page \*060]: Metric.

## Remove, Concept \*642 ‘-’ November 18, 2003

Replace is a Planguage verb indicating the removal of some specification from a defined set of specifications.

Keyed Icon [Remove \*642] ‘-‘ in context referring to a defined set of specs.

Related Concepts [Remove \*642]

• Replace \*641

• Add \*643

Synonyms [Remove \*642]

• Delete \*642

Type: Planguage Grammar

## Replace Concept \*641 November 18, 2003

Replace is a Planguage verb indicating the replacement of some specification with another.

For Example:

Security Performance [Project Axxx]:

Ambition: To have a highly competitive service capability for security administration and entitlement reporting related work processes, and work processes that involve security, both with respect to our employee effort to learn and do, and our customers effort to learn and do.

Scope: Account Opening and Entitlement Reporting.

Architecture: Compliance Implementation System?

Scale: Effort in individual real time minutes for a defined [Person] of defined [Capability] to successfully perform a defined [Task] under defined [Conditions, default: Normal Conditions].

Note: this strongly parameterized Scale is a basic structure for deriving Evolutionary steps of partial value delivery. These can be specified in for example Goal statements below.

End SP: Goal [Weak Case, Replace Capability = Expert, March 2004] 3 minutes???

Goal [Weak Case, January 2004] End SP x 2?? (about 10 minutes).

Weak Case: defined as: Person = Employee, Capability = Untrained Novice, Task = {Open User Account & Deposit Initial Amount}, Conditions = Outside Normal Office Hours

Example \*641: alternative using keyed icon for Replace (±):

End SP: Goal [Weak Case, ± Capability = Expert, March 2004] 3 minutes???

In the statement End SP, the qualifier ‘Weak Case’ is reused, but the normal ‘Capability = Untrained Novice’ scale variable (\*446) specification is replaced by ‘Capability = Expert’.

Related Concepts [\*641 Replace]:

• Remove, \*642 ‘-’

• Add, \*643 ‘+‘

Keyed Icon: [Replace \*641] ‘±’ (in context of specification strings, not numeric scalar points).

Keyed Icon Symbolism: ± minus indicates something is removed ( the’-‘) and something is added (the ‘+’) – i.e. replace.

Type: Planguage Grammar

## Report [SQC] Concept \*297 June 4, 2002

A ‘report’ is an *oral* statement made during SQC at a specification meeting or a process meeting.

Notes:

1. Reports during a specification meeting are issues, process improvement suggestions or ‘questions of intent.’ Reports during a process meeting are potential defect causes or potential process cures.

2. A report is made by a checker (an SQC team member) to get an item logged by the scribe, and to inform other team members of the issue.

3. Each report consists of a brief (‘seven words or less’) statement.

Example:

Issue: Line 230, Major, Rule 23. “An oral report when logged.”

Related Concepts [Report [SQC] \*297]:

• Issue \*276

• Process Improvement Suggestion \*088

• Question of Intent \*301

• Specification Meeting \*120

• Process Meeting \*119

Type [Report [SQC] \*297]: Feedback.

## Requirement Concept \*026 July 25 2010 (“-valued”), June 11, 2012 (Under stated conditions) = 3rd version after CE2005. Sept. 4 2012 (- future, + system)

A ‘requirement’ is a

**stakeholder-valued system state, under stated conditions.**

**Note September 4 2012**

1. The word **future** was deleted, since the *time* for the requirement is covered in the ‘stated conditions’
2. This advice was due to Kai Gilb, who pointed out that the required state could be past, present or future, not just future!
3. I added ‘system’ as it seems to make the *state* idea more intelligible, than having it implied. Kai also questioned the ‘state’ term and this is my response.

**Note 11 June 2012 TG**

1. I added the phrase “under stated conditions”
2. The ‘conditions’ can primarily be stated as a Planguage [qualifier set],
3. and in addition use any Planguage device that can attach to a requirement.
4. This includes the parameters such as Goal, Stretch, Fail.
5. It includes all other parameters that inform us of the requirement conditions, such as Sponsor, Version, Note, Issue, Risk and many others.

Some primary remarks on the definition:

• Requirements are used as an input to problem-solving (the problem being ‘how to reach the requirements’) processes; like design, architecture, strategy planning, more-detailed requirements, more technical requirements, and engineering.

• Requirements may be subject to ‘testing for existence’ in a system.

• Requirements exist without regard to the possible or actual means by which their future state is achieved.

• But a realistic requirement may be determined as a result of knowing the performance limits, and costs, of available designs.

• Requirements can exist even if no known means exist to satisfy them.

• A requirement will not necessarily be invested in, until its totality of complex priority information causes us to act on it.

• A requirement may exist, even when a formal requirement specification (\*508) does not.

Deeper Notes:

1. **Stakeholder** (\*233) specifically includes “any person, group or object, which has some direct or indirect interest in a system. The ‘object’ can include any documents or specification at any level, such as a law or a standards specification”

2. “**Stakeholder-valued”:** means that a given stakeholder, or set of them, can give a variety of value signals regarding the relative priority they place on achieving that future state. These priority signals\* can be analyzed in a given development environment, so that the actual implementation of the requirement is either delayed, implemented in some limited way, or never implemented. The requirement is finally ‘prioritized’ based both on several stakeholders’ several-value signals, plus other real time signals – for example a budget cut. There is no absolute certainty that a requirement will actually be implemented until it is actually implemented in practice. And there is no certainty that it will remain implemented once it has been implemented. Requirements are not absolutely ‘required’, they are conditional.

\* Priority signals: are of infinite variety, and may be analyzed in arbitrary sets, by any given analyst. They can include for example: deadlines, binary constraints, target levels, constraint levels, specification qualifiers, dependencies, risks, uncertainties, willingness to fund, formal authority, value, market willingness to pay, current and projected profitability, laws, regulations, and cultural values.

3. A requirement (\*026) is quite distinct from a ‘need’ (\*599): A ‘need’ is something desired by a defined stakeholder. The implication is that satisfying that need would have some value (\*269, ‘perceived benefit’ ) for some stakeholder. A need might not be agreed as a formal requirement, and it might not be prioritized such that it is actually acted upon (designed and implemented). ‘Need’ is a term often used as a stakeholder view of a problem before requirements specification is carried out.

4. Requirements are the baseline for deciding which design ideas (\*047) to use to satisfy them.

In practice, a set of requirements will be needed to reasonably optimize (discover, define, evaluate, prioritize) a practical set of designs needed to meet requirements.

5. “Future State”: refers to any observable (testable) attribute (like function, performance level, constraint compliance\*) of a defined system in the future. This does not imply that the system does not already, at present, already have the state, but just that it must (also) have it in, or by, a defined future time.

\* Constraint (\*218) compliance: can include compliance with design constraints, negative constraints, and positive constraints, as well as performance and resource scalar constraint levels.

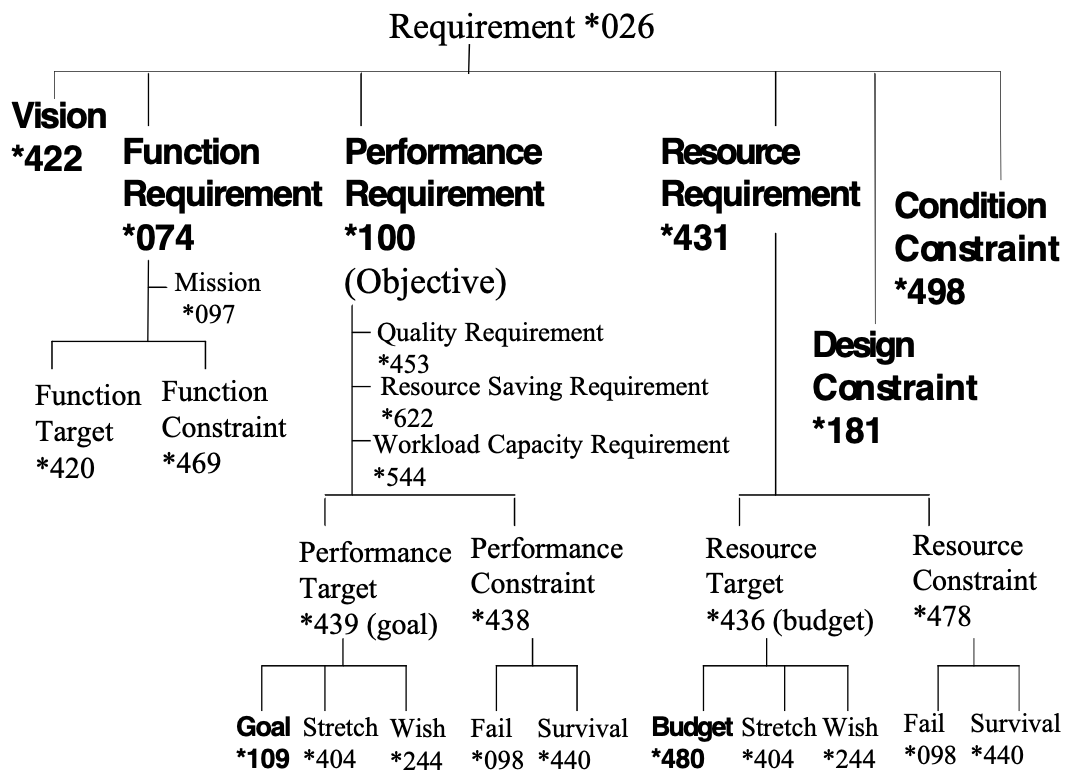


Figure \*026a: Requirement Concepts.

Mil\_Std 498:

Requirements are what the acquirer cares enough about to make conditions

for acceptance (may be "what" or "how")

Design is the set of decisions made by the developer in response to

requirements (may be "what" or "how")

(solutions plus decisions)

Pointed out by Niels Malotaux May 18 2005. He also helped simplify the main definition \*026.

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-95-MM-003].*

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

The IEEE 830:1998 definition, states

A requirement is:

1. A condition or capability needed by a user to solve a problem or

achieve an objective.

2. A condition or capability that must be met or possessed by a system

or system component to satisfy a contract, standard, specification, or

other formally imposed document.

3. A documented representation of a condition or capability as in items

1 or 2.

Notes on IEEE 830: Definition 3 is covered in Planguage by ‘Requirement Specification’ Concept \*508. The 2nd part is included due to the fact that the Planguage definition of stakeholder specifically includes inanimate things such as documents.

IEEE 830: Pointed Out by Erik Simmons, May 18 2005.

Other 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 understand the relative priority of, the most critical, or most profitable, stakeholder values.

3. A requirement specification is an input to a design process. Requirements give information to the designer about the necessary and satisfactory nature of their design. A design, whether a design specification or an actual design 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, as ‘input to’, or ‘output from’. For example, stakeholder value analysis, product line architecture, or component engineering processes.

5. Requirements can have different priority attributes. Priority is conveyed in a variety of ways, using a useful variety of priority signals, 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* (\*508). However, many documents, which have ‘requirements’ in their title, may contain little or no *real* requirements: it is unfortunately commonplace to find a high density of *design* specification (\*586) in so-called ‘requirements’, instead of their corresponding ‘real’ requirements. *A ‘password’ may be ‘required’, but no security defined as a basis*. Frequently, the most important, high-level and critical requirements are not stated clearly at all. <http://www.gilb.com/tiki-download_file.php?fileId=180>

You must be prepared to look for real and critical requirements in *other* documentation, and to ask questions of many stakeholders.

7. A design idea (\*047) 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 development process 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.

8. Alan Ashton-Jeanes has a particularly well thought out notion of requirement.

1. The people whose ends a system (\*145) serves are one class of stakeholders.  They value the actual results, or receive benefit (\*009)[[5]](#footnote-5).
2. The people who consider the future of a system and/or agree to the suggestions of others, are a different class of stakeholders.  They perceive potential value (\*269).
3. If these people take ideas forward (deciding to act or persuading others to decide to act, or considering the desirability of action), they can be described as “Founders[[6]](#footnote-6)” of the resultant system or course of action.
4. The arena within which action may occur is variously described as the “problem domain”, “scope” (\*419) or “context”[[7]](#footnote-7).
5. A perception of value within a particular context for action is a potential requirement.
6. A potential requirement coupled with authority (\*005) or discretion[[8]](#footnote-8) to pursue its satisfaction is a requirement. The person[[9]](#footnote-9) who confers the authority is the requirement’s “Sponsor”.
7. So, a requirement is a perception of value, in a particular context ,for action coupled with the authority to pursue[[10]](#footnote-10) its satisfaction: *the authorized and contextualized pursuit of value*.
8. The substance of a requirement (what is required) is the value. That of which it is a requirement is a future system.[[11]](#footnote-11) The context within which it is a requirement includes the project or endeavor (“development environment”) and the anticipated operating environment. Explicit or implicit authority to pursue the requirement (within a particular context[[12]](#footnote-12)) is a status of the requirement (in that context).

Source: Email Oct 10 2006, revised Feb 14 2007. [ashton.jeanes@googlemail.com](mailto:ashton.jeanes@googlemail.com)

9. I often use the term ‘objective’ (\*100) to express a high level of performance requirement. For example, ‘Usability’, which is often a complex requirement (consisting of 5 to 10 sub-requirements, like Intelligibility and Intuitiveness). Admittedly I tend to use ‘requirements’ more as a term when speaking to engineers, and ‘objectives’ when dealing with managers.

Example:

Usability:

Type: Quality Requirement.

Scale: Time to learn a defined [Task].

Goal [By Initial Delivery, Task = 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.

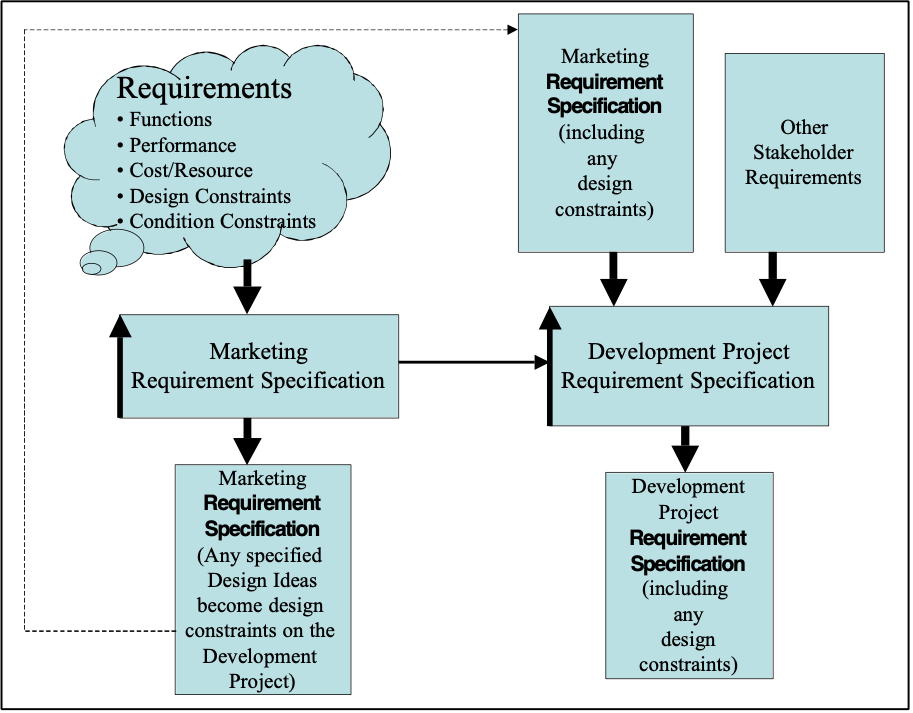


Figure \*026b: Design ideas from earlier processes, once ‘fixed’ (not subject to later change) become ‘design constraints’ (a type of requirement), at the next level of refinement (in an organization, or in the systems engineering process. Ideally, stakeholders will specify the requirements in their specific area of expertise.

Keyed Icon [Requirement \*026]: [@] “A combination of target and constraint.”

This is a generic icon. It is not used much in practice. In practice we use the specific icons such as Goal or Budget ( O----**>**-----> ).

Related Concepts [Requirement \*026]:

• Requirement Specification \*508: A written artifact containing a set of requirements, which may include additional background information.

• Need \*599

• Problem Definition \*598

• Problem \*270

• Design Problem \*654

• Target \*048

• Constraint \*218

• Stakeholder \*233

• Objective \*100: An ‘objective’ is not a pure synonym for ‘requirement.’ Objectives are performance requirements.

• Value \*269

• Benefit \*009

• Client Stakeholder \*650

• Server Stakeholder \*651

• Priority \*112

• State \*174

Type [Requirement \*026]: System Specification.

## Requirements Engineering Concept \*614 July 17, 2003

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 [Requirements Engineering \*614]:

• Requirement Engineering \*614

Related Concepts [Requirements Engineering \*614]:

• Requirement \*026

• Requirement Specification (Process) \*634

Type [Requirements Engineering \*614]: Discipline [Systems Engineering].

## Requirements Process Concept \*612 May 6, 2003

The requirements process 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

Synonym [Requirements Process \*612]:

• Requirements Engineering (process)

Related Concepts [Requirements Process \*612]:

• Requirements \*026

• Requirement Specification (noun) \*508

• Specification \*137

Type[Requirements Process \*612]: process

## Requirement Specification Concept \*508 March 4, 2004

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

Notes:

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

Synonyms [Requirement Specification \*508]:

• Requirements Specification \*508

Related Concepts [Requirement Specification \*508]:

• Requirement \*026

• Specification \*137

• Commentary \*632

• Background \*507

• Core Specification \*633

• Requirement Specification (Process) \*634

• Requirements Engineering \*614

Type [Requirement Specification \*508]: Specification.

## Requirements Specification Process \*634 May 26, 2003

The process of specification of requirements.

## Residual. Concept \*359 November 18, 2002

The remaining distance to a target level from a benchmark or current level.

Synonym: Residue \*359, Gap \*359

## Resolution. Concept \*525 November 18, 2002

The solution of a problem is a resolution.

## Resource Concept \*199 May 19, 2003

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 level, or a constraint level. Previous levels of resource utilization (incurred 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 type).

2. 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’.

Figure \*199 A: Arbitrary examples of some system resources.

3. 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.’

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 of policy:

Innovation Constraint [Division A]:

Type: Constraint [Financial Resource].

Definition: Total expenditure on new development cannot exceed 20% of the annual research budget for this division.

5. Resources that are input to a system differ from resource savings (a performance attribute type). Resources consumed are the costs of developing and operating a system. A resource saving is usually 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.

Figure \*199 B. Resources needed to describe a system (upper left arrows) are distinguished from resources needed to feed a process (lower left set of arrows). Both of these are different from resource savings (a system performance category) top right middle arrow.

LB I CANNOT EXPLIAN THE VERTICAL CONNECTION BETWEEN THE FUNCTION AND THE DESIGN? A design impacts the system generally , not the function locally, unless it is a function design. <-TG May 19, 2003

Synonyms [Resource \*199]:

• Fuel: Informal use only

Related Concepts [Resource \*199]:

• Resource Requirement \*431

• Resource Target \*436

• Resource Constraint \*478

• Benchmark \*007

• Target \*489

• Constraint \*218

• Cost \*033

• Resource Saving \*429

Keyed Icon [Resource \*199]: --->O “A scalar attribute arrow into a function oval.”

A simplified alternative is –O

Drawn icons [Resource \*199]: see examples above. Arrows into function oval for a system, and arrows into north side of a process icon.

Type [Resource \*199]: Attribute [Scalar].

## Resource Allocation: Concept \*503. October 29, 2001

The total quantity of a defined resource which has be allocated for defined purposes, at defined times, under defined conditions.

This is not just any budget, but some larger allocation for a company, division, project or phase from which individual budgets specifications may be made. It is the Limit for the sum of all budgets in a specified area and time.

Synonyms:

• Total Allocated Resource. \*503

• Cumulative Allocated Resource: \*503

• Total Budget \*503

Application:

• the difference between Resource Assumption (\*500) or the Resource Observed (\*502) and Resource Allocation (\*503) can be used to determine the Resource Available (\*504)

## Resource Assumption: Concept \*500 October 29, 2001

A resource assumption is an estimate or guess about the total quantity of a defined resource which will be available under given conditions [when, where, if].

This is not the same as a specific resource allocation. But the assumption is a useful basis for deciding on the allocation.

Synonym:

• Assumed Resource \*500

## Resource Available: Concept \*504. October 29 2001.

The resource available is the remaining portion of the resource allocated that can be used to budget designs, tasks or Evo steps.

.

Synonyms:

• Available Resource(s) \*504

Related Concepts:

• Resource Allocated \*503

## Resource Budget Concept \*430 August 31, 2001 2335

A resource budget describes how much of a resource we plan to have available.

A resource budget is a target level (Goal, Wish, Stretch) of a specified resource, which is related to a specific system component, with defined qualifiers for ‘where’, ‘when’ and ‘events’. This implies that one resource requirement can have many resource budgets.

It is quite different from a Cost Budget, which is a plan for how much of a resource we will use.

*Note 1: The term (resource) ‘budget’ is preferred for resources, rather than the term (resource) ‘target’; which we prefer to use to describe performance requirements (performance targets). This is a matter of taste. It gives clearer signals about the nature of the specification.*

Examples:

Space Allocation:

Type: Resource Budget

Scale: Storage which must be available.

Goal [Dial Out Module, Full Usability Quality, Release 2] 2 Megabytes.

Note this plan is different from how much we plan to actually use of the allocated storage at any given time (A Cost Budget). This just tries to make sure that two megabytes are really available if we need them. It also says we cannot expect more.

Related Concepts[ Resource Budget \*430]::

• Resource Constraint (Limit, Fail) \*478

• Cost Budget : \*479

• Budget \*480

• Saving \*429

• Budget Constraints (Limit, Fail) \*421

• Budget Targets (Wish, Goal, Stretch) \*481

Usage: Resource budgets are intentionally set by planners in order to allocate limited quantities of some overall (system-wide, project scope) resource.

Rationale: Resource budgets are necessary to avoid unintentional resource excess in a sub-system. They give clear limitations to designers, engineers, planners, and architects. They enable project management to synchronize sub-project efforts towards a single total resource utilization budget.

Type: requirement type.

Domain: CE.requirements.budgets.resource.

*Example:*

Budget: Scale $ total project.

Limit [Requirements per Contract Date] $10 million <- Contract 5.4.3

Fail [By Contract Complete] $9 million. Rationale: to ensure Corp. profit level

Goal [Lifetime Warranty] $9.5 million. Rationale: allow for lawsuits and risks.

*Specifies a resource limit \*476 and two resource budgets \*430 {Fail, Goal}.*

## Resource Constraint Concept \*478 June 6, 2005tg

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. Resource constraint levels (Fail, Survival) tend to be much worse than the stakeholder-valued resource targets (Budget, Stretch, Wish).

3. 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 [Resource Constraint \*478]:

• 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.

Ill. \*478: Some examples of Resource Constraint specification

Keyed Icon [Resource Constraint \*478]:

For Survival: ----------[--------------------]----->O “Upper and lower survival constraint limits.”

For Fail: ---------------!--------->O

Type [Resource Constraint \*478]: Constraint.

## Resource Efficiency: Concept \*366. August 5, 2001 2334

Resource efficiency is a system attribute describing how well resources are utilized when the system produces output.

For a given system output level, better resource efficiency, means ‘using less resources’.

A generic scale of measure for resource efficiency.

Scale:

A defined [Unit of Resource: default = Engineering Hours]

[total or marginal\*\*]

for producing a defined [Product]

at defined [Levels of Quality Attributes];

and using no more than defined [ Levels of Other Resources],

under stated [Conditions].

Type: system attribute concept.

Domain: system.attributes.resources.effciency.

For example

“ product x used 50% more engineering hours than its successor product”.

That is a statement about relative resource efficiency.

\*\* By ‘total’ we mean all relevant overhead costs, including some allocation of the capital and operational costs of Platform Development.

By ‘marginal’ resources we mean the pure costs of the product development, excluding already sunk capital costs, such as for Platform Architecture and process.

## Resource Saving Concept \*429 April 16, 2003

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 [Resource Saving \*429]:

• Saving \*429

Related Concepts [Resource Saving \*429]:

• Performance \*434

• Quality \*125

• Workload Capacity \*459

• Resource Saving Requirement \*622

Type [Resource Saving \*429]: Performance Attribute.

## Resource Saving Requirement Concept \*622 May 10, 2003

A resource saving requirement is a ‘resource reduction’ performance requirement. It specifies the specific savings in resource utilization that are needed: ‘how much’ is to be saved.

The focus is that it is, the specific level of *saving* that is the requirement (as opposed to the improvement in quality or workload capacity being the driving motivation).

A resource saving requirement is expressed as either:

• a performance level (a target or constraint level) compared to a defined benchmark level for a resource-related concept

Example:

X:

Type: Resource Saving Requirement.

Scale: Minutes.

Past: 30, Goal: 20.

Or as

• a ratio defined in a direct ‘saving’ scale of measure.

Example:

Y:

Scale: Percentage Relative Cost Reduction comparing Old Cost to New Cost.

Goal: 20% reduction. “A 20% saving.”

Notes:

1. The Scales used for Resource Saving Requirements must concern resource utilization: that is, a resource saving requirement is specified only if the target or constraint is defined in terms of resources (for example, as Maintenance *Cost* as opposed to Maintenance *Effectiveness*) needed.

2. A resource saving requirement is about a desired recurrent event (a resource saving) in the future, in connection with some current expenditure of resource (cost). For example:

• cost to manufacture

• cost to hire a new employee

• cost to repair a system

• cost to do a form of service

3. A resource saving requirement is not about saving resources needed to do the current project, as expressed in the resource budgets for the project (---->>---->O). It is about developing a system capable of saving resources when applied.

4. Performance requirements are not really *absolutely* different in their types (quality, resource savings, workload capacity). And these are not the *only* possible types of classification, which might be interesting. There is no critical consequence of this classification. It is quite possible that some objectives are mixtures (resource saving requirements and quality requirements) and there is no harm in declaring that they are both. So, if the decision-making about exactly which type an attribute actually is, gets too difficult, don’t worry! Use your best feeling and even allow mixed classifications. Or, invent your own useful classifications for your particular domain.

For example:

• ‘Ease of Learning’, such as ‘Scale: Time to learn a defined Task’, a usability concept, is a ‘quality’ value, not a resource saving, even if the quality goals are better than the benchmarks (in other words, an ‘improvement’).

• ‘Time needed to Train’ is a workload capacity attribute of a system. But, if it is specified with a resource Scale (space, time, money, data, effort, materials) and the benchmark value is less than the target value; then we have specified a resource saving requirement. We can make our intent clear by specifying Type: Resource Saving Requirement.

Example:

Manufacturing Labor Cost Savings:

Type: Resource Saving Requirement.

Scale: Human Effort needed for each defined [Product] manufactured in hours taken for each Product.

Past [Old Product]: 300 hours/Product.

Goal [New Product]: 100 hours/Product.

Note: Implied savings is 200 hours/Product.

Impacted By: Production Process Design.

Impacts: Product Cost -> Pricing -> Competitiveness.

Note the strong relationship between these two concepts (Ease of Learning: quality and Time Needed to Train: workload capacity) does not mean they are identical! The ease of learning is a *general potential* in the system itself (a quality). The time needed to train an employee is partly due to the ease of learning of the system itself, *but is finally determined by other factors* such as the individual learning ability, interruptions and motivation.

Rationale [Resource Saving Requirement \*622]: Why establish this classification of ‘Resource Saving Requirement’? I developed this concept (Resource Saving Requirement) as a separate category of performance requirement to provide a chance to increase the clarity of purpose.

With resource saving requirements, the intent to save is the ‘main idea’ behind the specification of the requirement. It is not that a specific level of performance has to be exhibited – it is that a *certain level of saving* has to be achieved.

Synonyms [Resource Saving Requirement \*622]:

• Saving Requirement\*622

Keyed Icon [Resource Saving Requirement \*622]:

See icons for Performance Requirement \*100, Performance Target \*439 or Performance Constraint \*438, as applicable.

Related Concepts [Resource Saving Requirement \*622]:

• Resource Saving \*429

• Performance Requirement (Objective) \*100

• Resource \*199

Type [Resource Saving Requirement \*622]: Performance Requirement.

## Resource Survival limit: Concept \*476 April 2, 2002

A Survival limit (\*440) on a scalar resource use.

Related Concepts:

• Survival limit \*440

• Performance Limit \*477

Type: scalar resource constraint.

*Example:*

Budget:

Scale $ total project.

Survival limit [Requirements per Contract Date] $10 million <- Contract 5.4.3

Fail [By Contract Complete] $9 million. Rationale: to ensure Corp. profit level

Goal [Lifetime Warranty] $9.5 million. Rationale: allow for lawsuits and risks.

## Resource Observation: Concept \*502. October 29, 2001

A resource observation is a determination, by any means, of the quantity of a defined resource that is available, at a given time, location and conditions.

Synonym:

• Observed Resource: \*502

## Resource Requirement Concept \*431 March 19, 2003

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.

Examples:

Maintenance Expenditure:

Scale: Million € annually.

Budget [First Four Years Average]: 3 million €.

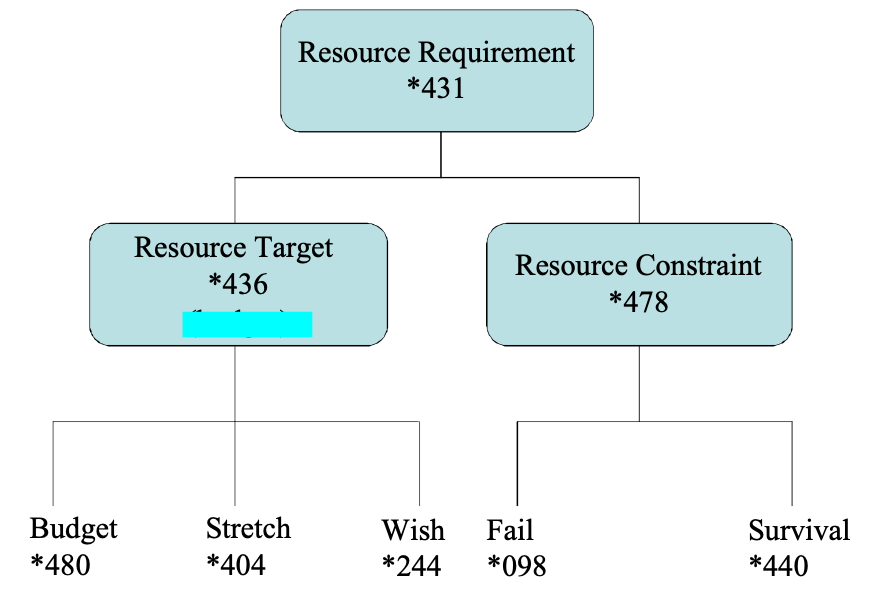
Fail [Any Single Operational Year]: 4 million €. <- Client limit in contract §6.8.

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.

A resource target and a resource constraint are specified for each resource requirement.

Related concepts.

Related Concepts [Resource Requirement \*431]:

• Resource \*199

• Resource Target \*436

• Resource Constraint \*478

Type [Resource Requirement \*431]: Requirement.

## Resource Target Concept \*436 March 22, 2003

A resource target is a qualified resource level we aim, or might possibly aim, towards keeping within, while working towards performance goal levels.

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 of 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.

Resource targets 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!

Related Concepts [Resource Target \*436]:

• Budget \*480

• Resource \*199

• Target \*048

• Wish \*244

• Stretch \*404

• Resource Constraint \*478

• Resource Requirement \*431

• Requirement \*026

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.

Some examples of Resource Target specification

Type [Resource Target \*436]: Target [Scalar].

## Resource Type Constraint "Resource Constraint" : Concept \*221. October 28, 2001

A resource type constraint puts *explicit* generic restriction "explicit restriction" on use of resource types or classes to solve a problem.

It is a restriction (\*498) and not itself a resource ‘budget’.

*Type: requirement.constraint class concept.*

*Domain: CE.requirements.constraints.resources.*

*For example: a design or engineering policy like*

*"All designs shall be selected with a view to using the least-scarce alternative resource in both development and local long-term operation."*

*is a resource constraint. But it is not a resource limit (\*440) or resource budget (\*430).*

*Notice that the example above impacts multiple resources, and is dynamic (meaning; ‘ its ininterpretation changes as the resource situation changes’).*

*Related Concepts [*Resource Type Constraint, \*221] *:*

*Resource Budget \*430, Resource \*199 , Cost Requirement, Cost Budget are all some type of Resource Constraint \*478, Restriction \*498.*

## Responsible: Concept \*646 19 April 2004

The person or instance assigned formal responsibility for achieving a specified plan element.

Synonyms [\*646] Responsibility, Result Responsibility.

Parameter [\*646] Responsible.

Type:

## Restriction: Concept \*498.

Synonym for Condition Constraint\*498.

## Result Concept \*130 May 10, 2003

A result is an outcome for a system after (or the ‘testable effect of’) the ‘Do’ phase of any Plan Do Study Act (PDSA) cycle.

Results are analyzed in the ‘Study’ phase.

Within Evo, ‘result’ is specifically used to refer to any outcome of a delivery cycle within a result cycle.

Notes:

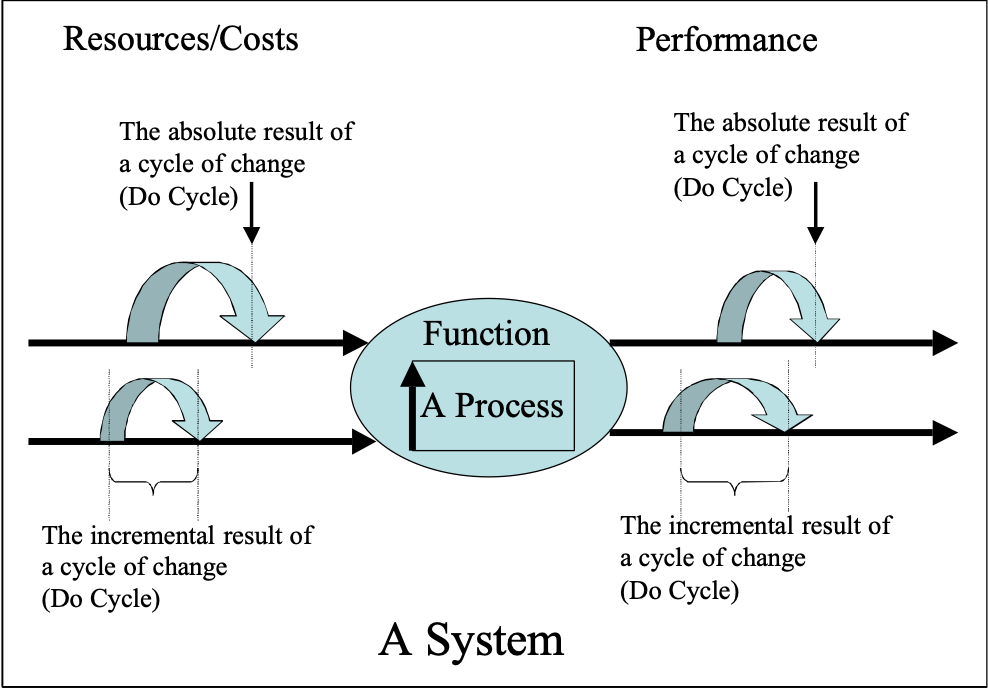
1. Results can be the outcome of many simultaneous factors. These factors include processes, events, and inputs that are both internal and/or external to a given system.

2. A result can consist of several impacts on the requirements; that is, several performance, resource and function attributes can be impacted as a result of a system change.

3. Results can be observed by testing system functionality and measuring system attribute levels (potentially any performance and/or resource levels may be changed).

4. A scalar result can be viewed in terms either of the *absolute level* or as an *incremental change* relative to a selected benchmark (See Figure \*130).

Figure \*130: Showing the distinction between an incremental and an absolute scalar result.



Synonyms [Result \*130]:

• Step Result \*130

Related Concepts [Result \*130]:

• Plan Do Study Act (PDSA) Cycle \*168

• Evolutionary Project Management \*355

• Delivery Cycle \*049

Type [Result \*130]: Process Output Data.

## Result Constraint "result constraint" : Concept \*200. August 7, 2001 0117

A term for a requirements constraint used for any of the requirement types {quality, cost or Function}.

*Rationale: This term is used where we do not wish to be more specific about the specific type of constraint. That means ‘result’ refers to any combination of performance, cost or function results.*

Example:

Top Quality:

Type: Result Constraint.

All market-critical qualities shall be planned, and tested, to be significantly, or at least noticeably-by-customer, better than any future co-existing competitive product, which is in approximately the same or lower price class.

*Note: not all constraints are result constraints, for example legal constraints and cultural constraints are not specifically result associated.*

*Type: constraint requirement class.*

*Domain: engineering.requirements.constraints.result constraint.*

## Result Cycle Concept \*122. January 3, 2003

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

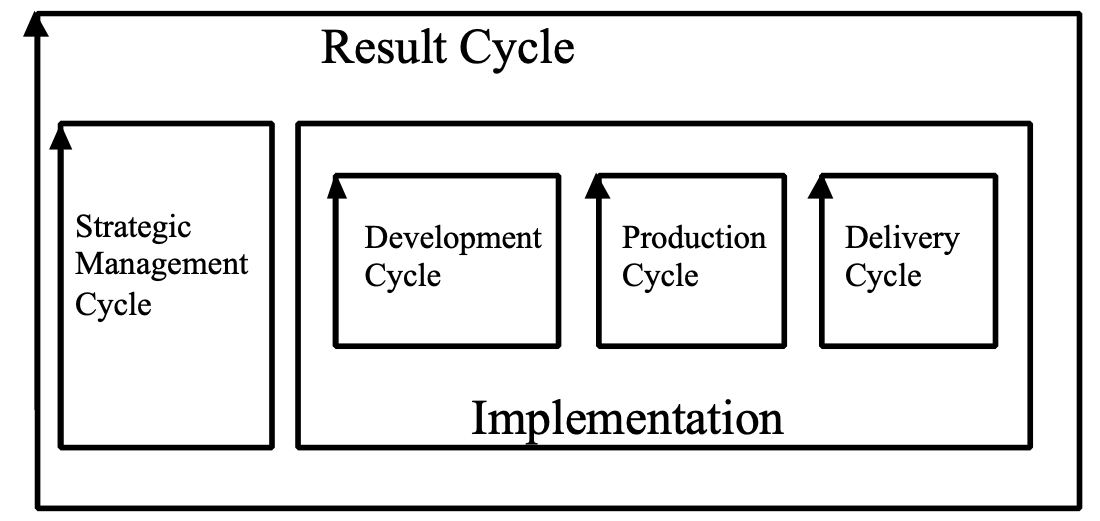
Notes:

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*.

Figure G.x: Diagram shows the component cycles of a Result Cycle.



4. A result cycle consists of:

• a strategic management cycle

• a development cycle (any development required or acquisition of the deliverables)

• a production cycle (any product integration or manufacture and distribution required)

• a delivery cycle (the actual delivery of the deliverable to the user)

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 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 Cycle \*122]:

• Result Production Cycle

• Step Cycle

Type: Process.

## Reuse Percentage: Concept \*367. August 7, 2001 0132

The [defined type of; default = % of volume] percentage of the product development which is accomplished by exploiting the existence of existing component’s design-or-development effort.

Rationale: This is a scale of measure that expresses the degree to which the development process succeeded in ‘not re-inventing the wheel’.

Hewlett Packard for example speaks of a ‘target reuse percentage’ [JANDOUREK96, p.61] where the ‘Platform’ “will provide Y% of the final product code”. The ‘platform’ being, among other thing’s, reusable software or firmware, made primarily for the purpose of enabling such reuse.

The costs of reusing can be arbitrarily high, to the point of being higher than not using them. So, this is not a measure of whether the reuse was a sound economic investment.

But the assumption is that we will attempt to win effort and time by reusing components, tools, processes, test plans and anything that pays off. HP speaks of reuse % varying between 10% and near 90% when using the Platform Development method [ibid.].

Type: product development efficiency metric.

Domain: CE.design strategies.metrics

## Review Concept \*197 May 25, 2003

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

Figure \*197: The diagram shows Review following SQC processes.

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 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 [Review \*197]:

• Specification Quality Control (SQC) \*051

• Specification Rules \*129

• Specification Review Rules \*543

Type [Review \*197]: Quality Assurance Process.

## Review Criteria Concept \*541 November 21, 2002

Review Criteria are the rules (and related checklists), the defined procedures, and any other standards or guidelines to be used to define and direct the analysis process, for a particular review activity. They are the questions the reviewers must ask and answer. They are the basis for determining the outcome of the review.

Related Concepts [Review Criteria, \*541]

• Clarity Rules (\*129) are an example of a standard for how well written a specification should be. They are used in Clarity Reviews \*607

• Design Rules, \*593 are rules for how to design, as opposed to how to specify a design (\*129) , it is a type of Content Rule.

Example: SR1: All quality concepts must be defined quantitatively with scales and levels.

• Content Rules (\*543) are a different type of standard since they focus on how good the ideas are in terms of meeting requirements, no matter how well they are specified in terms of completeness or intelligibility.

Example: CC1: All critical qualities for defined stakeholders must be specified at levels validated by stakeholders themselves.

Type: Standards

Domain: Reviews

## Review Focus Concept \*608 November 21, 2002

A Specification Review Focus is the primary purpose of a review. It is in practice defined by the set of rules, checklists, process description, people participation, or other devices for analyzing particular aspects of a specification.

If the Spec QC process is used, then the primary tool for determining review focus is the set of rules that are used to detect defects in the spec.

Examples of review focus:

• product profitability

• requirement clarity

• architecture feasibility

Related Concepts [Review Focus, \*608]:

• Specification Rules \*609 are one device to determine the review focus, by choosing a particular set of rules.

• Checklists \*016 checklists, is an additional tactic for sharpening the focus within a given set of rules.

• Spec Content Review (SCR) \*542 a content review is one general class of review focus

• Clarity review \*607 clarity review is another class of review focus which asks ‘is the spec clear to readership’ (even if it maybe is a bad idea to do).

• Review Criteria \*541 are the many types of standards (rules being only one) used to define and direct the analysis process for a particular review activity.

Type: review standards

Domain: reviews

## Rework Concept \*390 November 21, 2002

In any phase of engineering or management work, the human effort needed to correct work already carried out, due to the defects injected.

Usage: Rework is as much as 40%-60% of all systems engineering work, but it can be reduced to under 5% over several years by training, motivation and process improvement; for example, Raytheon [DION95].

Type: Metric.

Domain: Quality Assurance.

## Risk Concept \*309 August 21 2014, February 25, 2003**[[13]](#footnote-13)**

Risk is the **possibility** of Damage occurring.

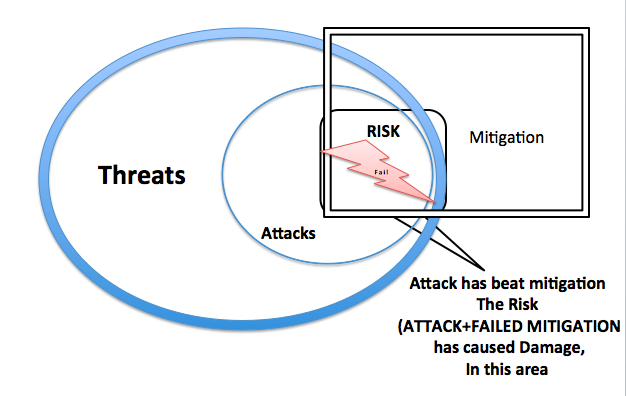
The **degree** of Risk is determined as a combination of an Attack and a Mitigation.

The higher the probability (frequency) of Attack, the higher the Damage. The higher the probability that Mitigation succeeds against the Attack, the lower the total Damage.

If there is zero (No) Attack, or if the Mitigation is 100% effective, then there is NO possibility of Damage Occurring. No ‘Risk’.

A Risk is something that has a positive probability of negative consequences or ‘Damage’.

A Planguage Risk parameter can be used to specify identifiable risks.



Notes:

1.Negative results are results that are worse than required, planned, or expected

2 The 'risk' of damage is not the same as actual damage. 'Risk' carries connotations of possibility and uncertainty.

Matthew Leitch

Tutor, researcher, author, consultant

[www.WorkingInUncertainty.co.uk](http://www.workinginuncertainty.co.uk/)

[matthew@workinginuncertainty.co.uk](mailto:matthew@workinginuncertainty.co.uk)

3. On calculating Risk of Damage.

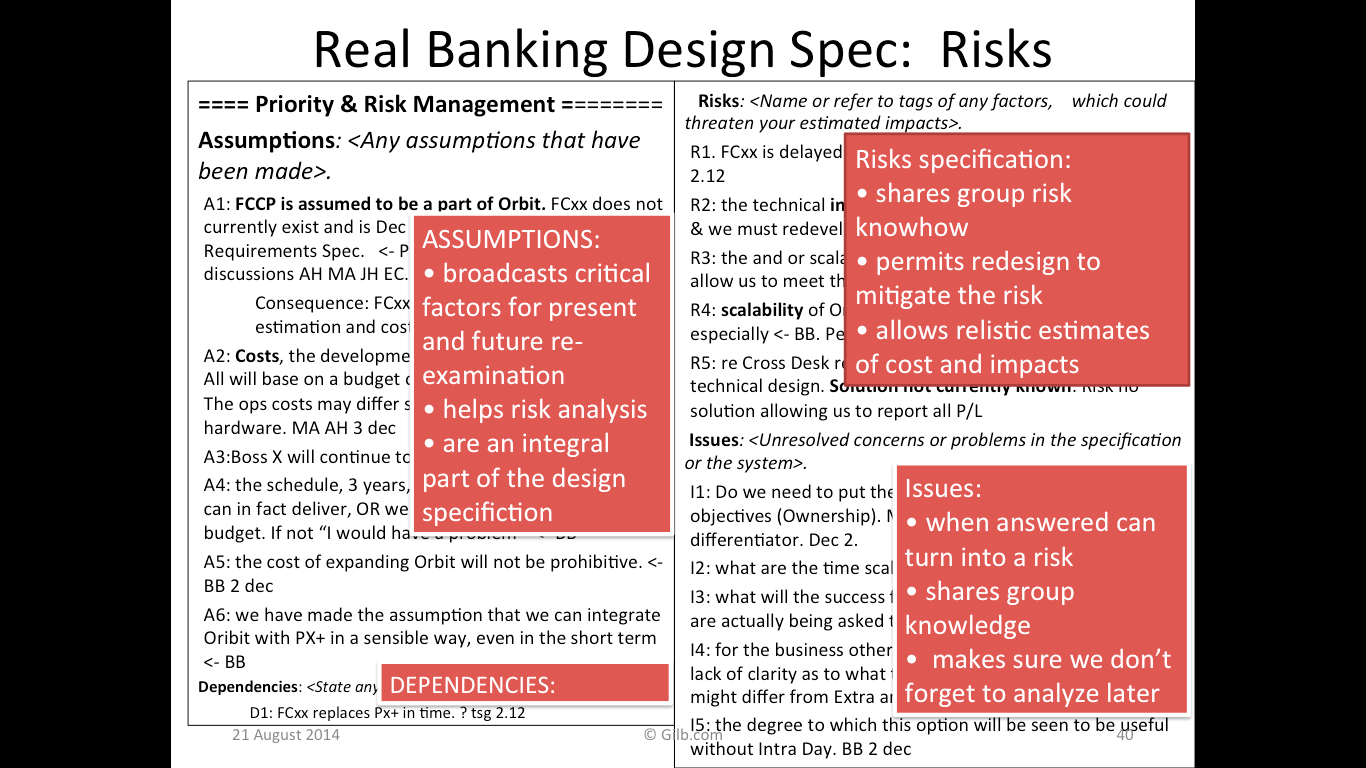
If 100% of Threats become Attacks, and Mitigation against these attacks is 100% effective, then there is 0% risk of Damage.

If 50% of Threats become Attacks, and Attack Mitigation is 50% effective, then 25% of Threats will cause damage.

Examples of risk factors include:

•  lack of information about a design idea

• inappropriate information about a design idea



Planguage Case \*309: The 4 concepts Assumption, Dependencies, Risks, and Issues are used when analyzing a design, to trigger the identification of ideas or instances, that may need further analysis (what are consequences, mitigations, frequency etc.). It is in practice immaterial to us which category is selected to note the concern, or even if the categories are ‘correct’. The important idea is that we have triggered people to recognize, and specify for further consideration, things that potentially need more planning. (added 21Aug2014)

Source

1. Slides Banking Experiences
2. Real client work done at a London International Bank, using the Competitive Engineering book Design template.

Source: See Bernstein’s book on the history of risk. One prominent economist (Knight) wanted to distinguish risk from uncertainty, in the sense that risk was measurable [BERNSTEIN96, 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 [Risk \*309]:

• Negative Consequences ??

Negative Results??

Probability of Attack Damage (8.2014) working on this

Related Concepts [Risk \*309]:

Threat \*309

Mitigation \*670

Attack \*672

Damage \*673

Issue \*276

Dependency \*189

Assumption \*002

• Uncertainty \*310: Uncertainty does not imply a negative result as clearly as ‘risk’ does. Uncertainty means the result could be positive, negative or neutral.

• Safety Factor \*131

Type [Risk \*309]: Noun, Parameter [Risk Analysis].

## Robustness Concept \*673 140814

Robustness is a system attribute that defines how well it resists Attacks; which is a function of its Mitigation capability (such as Security Architecture).

The less Damage from a given Attack (type and frequency) the more Robust it is.

Another view is that the more Attacks of a given type it can Cope with: {Detect, Rebuff, Recover} the more Robust it is.

Note:

System Integrity, in Security situations is a special case of the more general Robustness.

Related Concepts [Robustness \*673]:

• Damage \*673

• Threat: \*671.

• Attack \*672

• Mitigation\*670

Synonym [Robustness \*673]

Integrity (see Integrity in CE Book Ch.5)

Type [Robustness \*673]: System Attribute

## Role Concept \*253 February 27, 2003

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

Related Concepts [Role \*253]:

• Role [SQC] \*411: Role as used in Specification Quality Control (SQC).

• Stakeholder \*233

Type [Role \*253]: Organizational Concept.

## Role [SQC] Concept \*411 February 27, 2003

Within SQC, a role is a specialized checking responsibility, assigned to an individual checker (and specified in the master plan) by the team leader.

Combinations of roles can be assigned to a single checker, and checkers can have overlapping roles, as purpose dictates.

Rationale: Roles are assigned with the objective of increasing both an individual’s productivity, and the SQC team's productivity. Productivity in SQC is measured by the number of major issues per page found by the team (effectiveness). Roles can however be assigned to support any current, local objective of the SQC process (for example, training). Effective assignment of roles permits each checker to find a relatively large number of unique issues. It can also enable a larger volume in total of documentation to be checked during the SQC (as checkers can be role-assigned to use different sets of documentation).

Notes:

1. Roles ensure the documentation is checked from several different perspectives. Roles can be roughly classified as follows:

• stakeholder – using a checklist, check from a particular stakeholder viewpoint (like ‘tester’)

• document or territory – ‘where’ the checking shall take place

• defect type – what kinds of defects are we to especially look for (use rules and checklists)

The team leader can plan the checking to concentrate on specific stakeholder views (stakeholder roles) and/or specific documents (document roles). The team leader might first assign stakeholder view roles with their associated rules. They then might decide which documents to assign to each role. Finally, they might consider whether any of the documents require special attention and assign additional document roles (and rules) accordingly. Examples of stakeholder view roles are user, tester, designer, maintainer, legal advisor and marketing. An example of a document role would be to spend more effort on checking that the marketing plan, a source document, is consistent with the rules.

2. Roles are for use during the Checking sub-process and are agreed during Kickoff. In the Specification Meeting, if ‘double checking’ is carried out during the specification meeting, roles would again apply and might be intentionally modified.

3. A checker can modify their own assigned roles, as needs and reason dictate, during the checking processes.

4. The sequence of planning, and exact role definitions, are completely open to the imagination of the team leader while planning, in an effort to meet their SQC objectives.

5. To help interpret the rules, there are checklists for individual roles to assist checkers to search effectively for their specified classes of major defects.

Related Concepts [Role [SQC] \*411]:

• Role \*253

• Rule \*609

• Checklist \*016

Type [Role [SQC] \*411]: Organizational Concept.

## Root Causes Concept \*263 November 21, 2002

Root causes are the ultimate upstream reason why an error/defect/fault/malfunction sequence occurs.

Rationale: Root causes are interesting because by identifying them you will generally have identified the most efficient points for change to prevent downstream problems.

Related Concepts:

• Common Cause (\*020):

• Special Cause (\*019):

• Defect Prevention Process (DPP) (\*041)

• Process Improvement (\*114)

• Statistical Process Control (\*466)

• Continuous Process Improvement \*424

• Systemic \*262

Type: Continuous process Improvement concept

Domain: Continuous Process Improvement.

## Rule Concept \*333 May 25, 2003

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

Notes:

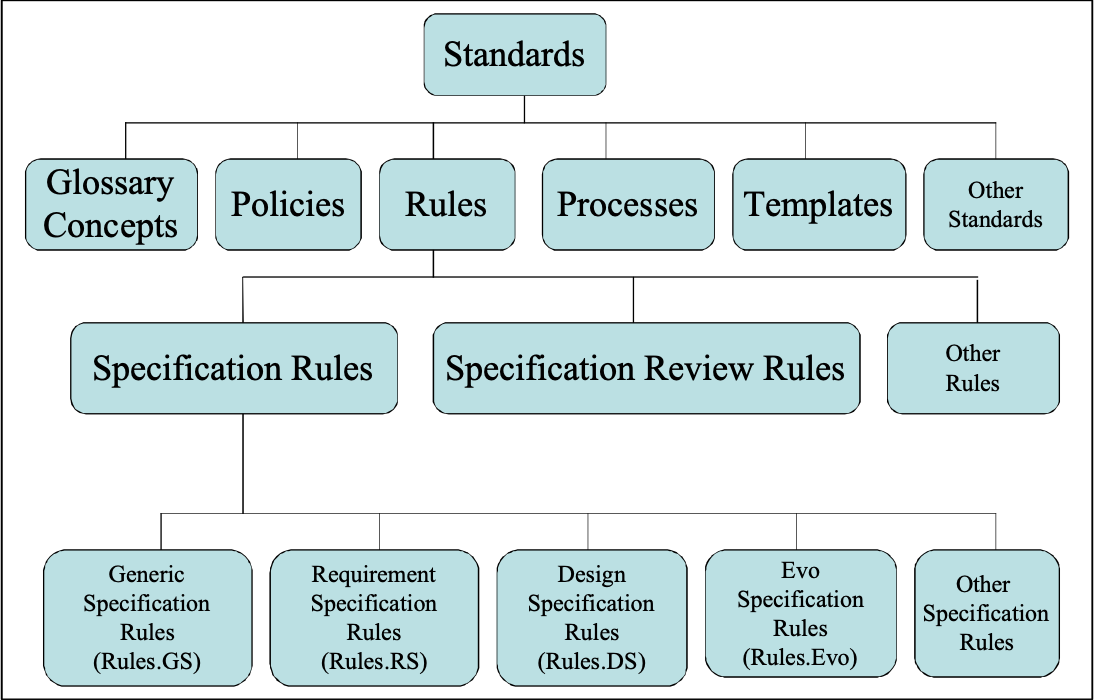
1. Numerous different types of rule exist, including business rules, system rules and design rules.

Related Concepts [Rule \*333]:

• Standard \*138

• Specification Rule \*129

• Specification Review Rule \*543

Figure \*333: Rules as standards. Some of the different types of Specification Rules are shown.

Type [Rule \*333]: Standard.

## Safety Deviation Concept \*405 June 16, 2003

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.

Figure \*405: Diagram showing calculation of a safety deviation for a performance attribute. 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 150%, it must be 50% beyond the target level. The distance (x) is worked out from the distance between the target and the baseline (2x).

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. A Safety Deviation is measured as a percentage difference between the estimated or actual percentage impact and the safety margin. Percentage values are usually used. It could however be expressed as the difference between the two levels using the actual units of measure.

Safety Deviation is calculation in a slightly different way for cost attributes, than for performance attributes. because the value for the Safety Margin is calculated in a different way.

Assuming percentage values are being used,

The calculation of safety deviation for *performance attributes* should be as follows:

(Percentage Impact Estimate or Percentage Actual Measure) – Safety Margin

while the calculation of safety deviation for *cost attributes* should be as follows:

Safety Margin - (Percentage Impact Estimate or Percentage Actual Measure)

5. 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 Deviation \*405]:

• Safety Factor \*131

• Safety Margin \*637

Keyed Icon [Safety Deviation \*405]: X.±

Type [Safety Deviation \*405]: Metric.

## Safety Factor Concept \*131 June 9, 2005

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.

Keyed Icon [Safety Factor \*131]: nX “Where n is the numeric safety factor.”

Example:

Safety Factor: 3X.

Type [Safety Factor \*131]: Metric, Parameter.

## Safety Impact "safety margin" : Concept \*406. August 12, 2001 1622, 1721, 1804

The safety Impact is the % deviation from target levels (targets = 0% level).

Note 1. The safety impact is a measure that is independent of the safety factor required (if any). It is an indirect measure of how well that a required safety factor is met in theory or in practice. It can be compared to the safety factor to see how well we are doing in achieving safety factor levels.

Note 2: if the deviation is more than target for performance measures, and less than budget for cost budget targets then we have ‘a degree of safety’. If we are worse than the target levels then we have a ‘negative safety margin’.

For example:

•  ‘the safety impact is +50%’, means it is 50% better than the target level (100%).

•  If the safety factor is 2x for a performance target, then the safety impact of +50% is ‘good’,

• but it falls -50% (short) of a possible required level of ‘100% over the target’ (= 200% impact) which would be ‘insufficient’ over-design.

Usage:

• the ‘safety impact is +50%, for a performance goal’, means 50% over the target level. It means ‘50% more than required by the target itself’ (which is at 0% level), but perhaps ‘less than the requirement specified by a safety factor’.

• the ‘safety impact is +50%’ for a cost budget means that the estimated cost is 50% better ( -50% compared to budget level 0%) than it should be to meet the budget (which is 0% level). So it is half the target level.

Type: Risk Management measure.

Synonyms [Safety Impact]

•  Safety Impact Percentage (%),

• safety differential,

• benchmark safety margin

(as opposed to a required constraint safety margin, a generic constraint),

• estimated safety margin

Domain: CE.{design, project management}.Impact Estimation.impact analysis. Safety factor.Safety Impact

Related Concepts [Safety Impact \*406]:

• Safety Factor/Margin \*131

• Safety Deviation \*405

• Impact Estimation \*283

• Impact %: the impact in relation to the target (100%), which is 0% safety impact.

• Target levels

• Impact Estimate \*433

• Impact (noun) \*087: the effects of a design on attributes

• incremental impact \*220 The potential benefit or savings.

•  Incremental Scale Impact \*307. The increment expressed as a differential on the defined Scale.

Impacts (parameter) \*334

• Percentage Uncertainty \*383

• Percentage Impact \*307

• Absolute Impact \*403

• Incremental % Impact \*462. The increment expressed as a % differential in relation to the Target level expressed as 0% differential, =100% of target).

## Safety Margin Concept \*637 June 9, 2005 LB

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:

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).

A safety margin can also be expressed as a percentage based on the target or constraint level being 100%, and the baseline being 0%.

A ‘safety factor’, by comparison, is a dimensionless ratio.

Related Concepts: Safety Factor \*131; Safety Deviation \*405.

Figure shows the required safety margin for a required performance level.

Related Concepts: Safety Factor \*131; Safety Deviation \*405.

Type [Safety Margin \*637]: Metric.

## Sample [SQC, QC] Concept \*298 November 20, 2002

In a quality control process, a sample is a small part of a document, product or system, which is selected for checking.

Rationale: The intent of an SQC sample is that it will give some information about the specification, without the cost of checking more of it. The intent would never be to identify or remove all defects (even from the sample, and certainly not from the non-sampled rest of the document).

Note:

1. The observed effectiveness of the SQC process is in the range about 30% (or lower) to a maximum of 88%; meaning 70% to 12% of defects in the area sampled would not be discovered on a single pass of the SQC process. This means that even 100% QC of all parts of a specification will still only give us a sample of all the defects in the document.

2. Any form of testing can also use sampling.

Related Concepts:

• SQC \*051

• Testing

• Statistical Process Control

Type: statistical device

Domains: QA, SQC.

## Scalability Concept \*685 30 May 2018

Scalability is a ‘system quality’,

which includes different types\*\* of scaling

and different degrees of scaling

for defined stakeholders

using a defined set of scaling strategies

Notes: \*\* types of scaling include, but are not limited to:

volume, speed, backup activity, diagnosis activity, security activity.

See my slides and papers on Scaling:

**SCALE-FREE:**

**Practical Scaling Methods**

**for Industrial Systems Engineering”**

lecture slides

http://concepts.gilb.com/dl892

Tom Gilb

for ALSSE

Norsec, Oslo

Monday 12 September 2016

**“Beyond Scaling: Scale-free Principles for Agile Value Delivery - Agile Engineering”**

http://www.gilb.com/dl865 (Paper)

(Jan 8 2016)

**Scalability Metrics:**

**An Engineering Structure,**

**and**

**Principles, for an Agile Worl**d

for June 5 2018 DND Conferent

???? URL

## *Scalar Concept \*198 May 11, 2003*

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 [Scalar \*198]:

• Scale \*132

• Binary \*249

Keyed Icon [Scalar \*198]: ----------> “A scalar arrow. This icon is more

properly used to represent a scalar attribute, but it seemed reasonable to

use the same icon here, as it is the same underlying concept.”

Type [Scalar \*198]: Adjective.

## Scalar Baseline. Concept \*490. May 6, 2004

A scalar Baseline is a Baseline expressed on a defined scale or measure.

Related Concepts:

• Function Baseline \*489

• Benchmark \*007

• Baseline \*351

Edit note: May 6 2004: need to clean this up wrt baseline and benchmark. Is there a distinct concept Scalar Baseline? TG

## Scalar Constraint. Concept \*468. November 24, 2004.

A Scalar Constraint is some kind of limitation on the delivered level of a defined Scale of measure.

The following scalar constraint types exist:

• Tolerable Limit/Must Do Level/Worst Acceptable Case (\*539 >>) --->>---?????

• Failure Limit (\*098, !) ----!----

• Failure Range (\*546, !!) -----!!!!!!---

• Survival Limit (\*440, [, ]) ----[====]---

• Survival Range (\*585, between & including square brackets levels) [=====]

• Catastrophe Limit (Crash parameter, \*602, •)

• Catastrophe Range (\*603, •••••) O••••••[

• Safety Factor \*131 X

Type: constraint, requirement, scalar levels

Related Concepts [Scalar Constraint \*468]:

• Constraint \*218

• Binary Constraint \*467

• Performance Constraint \*438

• Budget Constraint \*421.

• Must Do Level (\*539, >>) --->>---

• Failure Limit (\*098, !) ----!----

• Failure Range (\*546, !!) -----!!!!!!---

• Survival Limit (\*440, [, ]) ----[====]---

• Survival Range (\*585, between & including square brackets levels) [=====]

• Catastrophe Limit (Crash parameter, \*602, •)

• Catastrophe Range (\*603, •••••) O••••••[

• Safety Factor \*131 X

Keyed icons [Scalar Constraint \*468] [ and ] and >> (Fail), and others as per above list.

## Scale Context \*640, September 2, 2003

A scale of measure context is all the additional information that applies to a scale of measure, in addition to the units of measure.

The context can be defined in the Scale parameter specification itself, or elsewhere. The context can be defined in qualifiers for any parameter referencing the defined scale (like Goal [Type = Man] 60%). It can of course be defined in any definition that is referenced in the Scale specification, or in anything referencing the Scale definition.

Related Concepts [Scale Context \*640]:

• Scale of Measure \*132. The scale context is one part of this.

• Units of Measure \*560. The scale context adds more practical specifics to this.

• Scale Variables \*446. A type of ‘scale context’.

Type [Scale Context \*640]: Metrics concept.

## Scalar Target Concept \*470 March 15, 2003

A Scalar Target is a performance target or resource target level specification.

Scalar Targets define stakeholder-valued levels of system performance and development costs.

Scalar Targets are not scalar constraints, like Fail; nor are they benchmark levels, like Past.

Scalar target parameters, in order of priority, are Goal/Budget, Stretch, Wish and Ideal.

Related Concepts [Scalar Target \*470]:

• Performance Target \*439

• Resource Target \*436

• Goal \*109 (O->->),

• Budget \*481 (->->O) ,

• Wish \*244 (>?), or

• Stretch \*404 (>+).

• Ideal \*328 (>++)

• Target \*048

• Scalar constraint \*468.

• Benchmark level \*007

------>Budget----->+Stretch----->?Wish --->O------>Goal----->+Stretch ----->?Wish--->

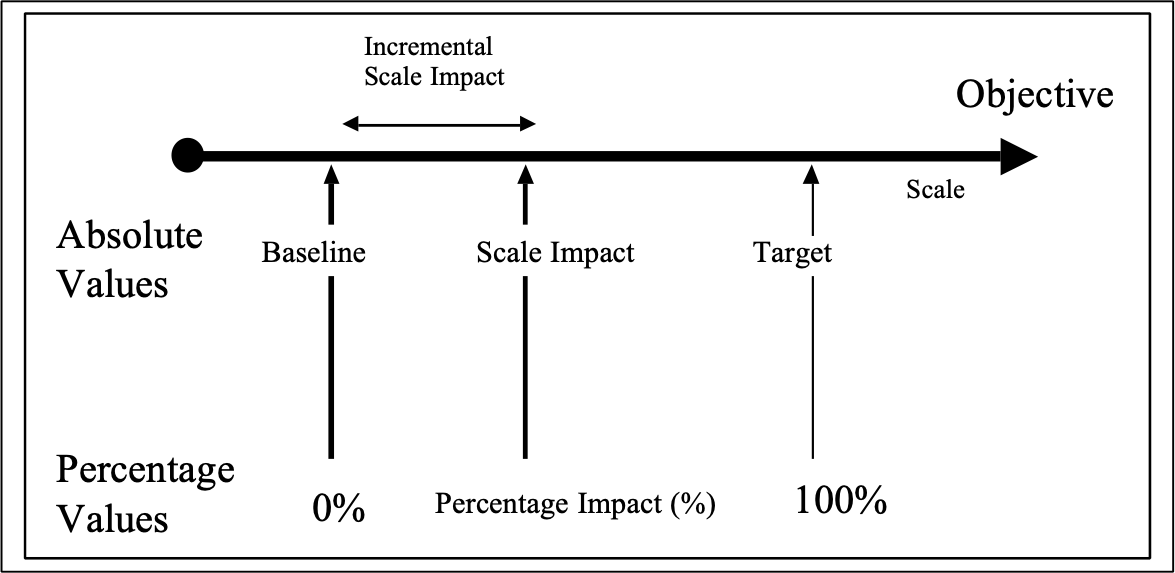
Fig. \*470 Keyed icons indicate some scalar targets.

Type [Scalar Target \*470]: Requirement Level, Target.

## Scale Impact Concept \*403 January 20, 2003

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.

In an Impact Estimation table, 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.



Note: 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.

Synonym [Scale Impact \*403]:

• Absolute Impact \*403

Keyed Icon [Scale Impact \*403]: **-|-|-.->**

“Made up from the symbols for Scale (-|-|-) and Impacts (->).”

Related Concepts [Scale Impact \*403]:

• Incremental Scale Impact \*307

• Percentage Impact \*306

Type [Scale Impact \*403]: Scalar Design Metric.

## Scale of Measure Concept \*132 August 17, 2004, Units Jan 24 2011

A scale of measure defines a single scalar attribute dimension.

It helps us ‘quantify’. It is the basis for quantifying variable attributes.

All scalar numeric level estimates, specifications, or measurements, are used with an implied (nearby and previous), or explicit, reference to a defined scale of measure.

A ‘Scale:’ parameter specification defines the units of measure, and includes any other useful context, including scale qualifiers (‘for defined [Tasks]’), normalizers (‘per week’), and environment specification (‘for Expert Hackers’).

Some elements of the *context* of a scale of measure, but never the units of measure themselves, may be specified *outside* the Scale specification; for example in target qualifiers, or in term definitions.

Notes:

1. A defined Scale is used as the basis for specifying numeric levels of an attribute. All the benchmark, target, constraint, and estimate parameters (re-) use the defined Scale as the reference for what is being *quantified*, that is, for the *units of measure*, and their *scale* *context*. There are likely to be many references to the same scale of measure, so we define it once, and ‘reuse’ the scale definition many times. A useful scale of measure will typically have 10-20 words or more, and will rarely be so simple as a one-word unit of measure such as ‘seconds’ (for what?).

2. A Scale must be conceptually distinguished from the many potential defined numeric levels, specified somewhere along it. In other words, a Scale is not a benchmark level (like a Past level), a target level (like a Goal level) or a constraint level (like a Fail level).

3. A Scale does not specify the measuring instrument: a measuring instrument is specified by a ‘Meter’ parameter. One or more Meter specifications can be associated directly with any one given Scale specification. There can be many different appropriate ways to measure along a given scale, for different conditions.

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 numeric value on a Scale. For example, a Meter is the ‘voltmeter’ for measuring on a scale of ‘volts’. A Scale is *conceptual*, while a Meter is a real world practical means of obtaining measurements of an attribute.

4. There is a big distinction between a scale of measure (Scale) and a measuring tool (Meter). Numeric levels, stated with their 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*.

5. Many Scales are specified as ‘generic scales.’ A generic scale is a Scale that includes ‘scale qualifiers’ in its definition. ‘Scale variables’ have to be assigned to the scale qualifiers whenever such a Scale is used in other parameter specifications, in order to have a precise parameter definition. Such scale variables can be set up by default within a Scale definition. See example below.

Note, also that *additional* qualifiers, not specified as scale qualifiers, can also be added, as required, to a *parameter* specification (like Goal), to define it more precisely. The parameter specification qualifier is not limited to defining the defined scale variables with corresponding scale qualifiers.

Example:

Scale: Time in seconds to Master defined [Task: Default = Update] by defined [Learner Type].

There are two scale qualifiers in the above generic scale definition, which require definition by scale variables. For example, ‘Goal [Task = Update, Learner Type = Novice]: 30%’.

6. Each elementary scalar attribute definition requires a single defined Scale. A complex attribute, when decomposed into a set of defined elementary attributes, will be represented by a set of Scales.

7. A good scale of measure will accurately reflect some stakeholders’ values. Bad scales will not. For example ‘mean time between failure’ reflects reliability for system users better than ‘bugs per 1,000 lines of code’.

“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. 1988. Dynamic Manufacturing. New York: . Page 77. Quoted in (Mintzberg 1994 Page 24).,

“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)

Related Concepts [Scale of Measure \*132]:

• Scale Context \*640

• Metric \*095

• Measure \*132

• Units of Measure \*560

• Scale Qualifier \*381: Scale qualifiers are terms within a scale definition, which need to be specifically defined by scale *variables*, when the scale definition is later used in scalar level specifications, like ‘Goal’.

Example:

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

‘Time Units’, ‘Task’ and ‘Employee Type’ are all scale qualifiers.

• Scale Variable \*446

Example:

Goal [Task = Update, People = Naval Officer, Places = At Sea]: 99%.

‘Update’, ‘Naval Officer’ and ‘At Sea’ are all scale variables assigned to the scale explicitly specified (Places =) scale qualifiers.

• Meter \*094: The specified measuring process, or test, to determine, for a real system, where we are on a defined scale of measure (Scale).

Example:

User Friendly:

Type: Quality Requirement.

Ambition: To consistently exceed Competitor’s ease of learning.

Scale: Time in seconds to Master a defined [Task] by defined [Learner Type].

Meter: <Use good academic practice, do at least 10 Tasks, with at least 5 Learner Types and at least 50 people>.

Record [Competitor AA, Product XYZ, Task = Dial Out, Learner Type = Novice]: 120 seconds <- Our current tests.

Goal [Our Company, Product ABC, Task = Dial Out, Learner Type = Novice]: < 10 seconds <- Marketing Requirement 4.5.7.

Master: Defined as: Ability to pass a suitable approved test.

The Scale includes two scale qualifiers (Task and Learner Type). These are reused as scale variables in the benchmark (Record) and target (Goal), in order to tailor the meaning to particular sets of Task and Learner Type. The scale variables are usually local to the specific specification (in this case, User Friendly).

Synonyms [Scale of Measure \*132]:

• Scale\*132 (short form), can be written ‘scale’ when not the parameter itself.

• Quantification Scale \*132 (suggested Feb 2004 Richard House).

• Measure \*132

Parameter [Scale of Measure \*132]: ‘Scale:’

• May be written ‘Scale of Measure:’ (long form of parameter).

Related Concepts [Scale of Measure \*132]:

• Units of Measure \*560 (core *part* of a Scale specification)

Keyed Icon [Scale of Measure \*132]: -|-|-

Type [Scale of Measure \*132]: Metrics Concept, Parameter.

## Scale Qualifier Concept \*381 February 24, 2005

is an optional term within the *definition* of a Scale of Measure.

It defines generic conditions (such as ‘Task’ or ‘Stakeholder’) that need to be specifically defined in other statements - that reference the Scale specification, such as Past or Goal.

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 Qualifier Definition (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 Qualifier Definition 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 signaled by enclosing it in (optional, but useful signal) square qualifier brackets. The word ‘defined’ is optionally specified immediately prior to the square brackets or scale qualifier term, to help emphasize that a more specific definition needs to be defined when the Scale is referenced, for example by a Goal statement.

Example:

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

Synonym [Scale Qualifier \*381]:

• Scale Parameter \*381

• Scale-of-measure specification parameter \*381

Related Concepts [Scale Qualifier \*381]:

• Qualifier \*124

• Scale Qualifier Definition \*446

Type [Scale Qualifier \*381]: Specification Concept.

## Scale Qualifier Definition Concept \* 446 July 29, 2004

A scale qualifier definition is the specific term assigned to ‘finally’ define a qualifier condition for a corresponding scale qualifier *(for example ‘Update’, in ‘Scale: time to complete a defined [Task]. Past [Task = Update] 30 minutes.’*

If we fail to explicitly define a scale qualifier definition in a particular statement, then a defined *default* scale qualifier definition (… for defined [Tasks, Default = Update]… ) might used. If there is no default defined, in that case, then that statement is defective (undefined).

Examples:

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 qualifier definition, out of the set of possible days of the week.

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 , using explicit references (Task =…) to the Scale Qualifier.

Goal [Task = Update, People = Naval Officer, Places = At Sea]: 20 minutes.

Update, Naval Officer, and At Sea are sequence implied scale qualifier definitions, defining one of the three scale qualifiers (Task, People and Places). The scale qualifier definitions 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?

Synonyms [Scale Qualifier Definition \*446]:

• Scale Qualifier Specification \*446

Related Concepts [Scale Qualifier Definition \*446]:

• Scale Qualifier \*381

• Specification Variable \*456

Type [Scale Qualifier Definition \*446]: Specification Variable.

## Scale Type \*639, September 2, 2003<- Stuart W

The scale type is a classification of the scale of measure definition. For example

• scale with scalar variables

Note this is a very rough draft. I need to work on it more.

## Scale Uncertainty Concept \*143 November 8, 2002

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 it may be useful to specify the asymmetric numbers (e g +30%, - 40%).

Keyed Icon: -|-|-.±? “Made up from the symbols for Scale and Uncertainty”.

Type: Design concept. Numeric Variable.

Domain: Impact Estimation.

Related Concepts: Scale Impact \*403.

## Scribe Concept \*410 November 13, 2002

The scribe writes up the editor advice log or other team notes at an SQC meeting.

Note: This can be any one of the team members. By default, the team leader will scribe. ‘Who scribes’ is not a critical decision.

Type: Role.

Domain: SQC.

## Scope Concept \*419 May 4, 2003 B

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’ [HOOKS01, pages 43-58].

• Local Scope: a local specification is unable and unwilling to influence or determine specifications (requirements, designs) beyond a defined sub-system area.

Related Concepts [Scope \*419]:

• View \*484

• Context \*483

• Qualifier \*124

• Time \*153

• Place \*107

• Event \*062

• Mission \*097

• Core Ideology \*664 (also Core Purpose \*662 and Core Values \*663)

Example:

Scope [Project X]: USA Parent Market only.

Example of an explicit Scope specification. Scope can otherwise be indicated by a qualifier and by many other parameters (such as Authority and Rationale for example).

Type [Scope \*419]: Parameter, Systems Engineering Concept.

## Serial Development Concept \*364 February 25, 2003

Serial development, in Planguage, is a development process involving development work on only one Evo step at any one time.

Rationale: The advantage is that the initial lessons of the previous project work are available to the next Evo step. The disadvantage is that the ability to rapidly produce related product variants for a demanding market is reduced, in terms of the “Time Between Successive Products” (TBSP). Also long lead time for any step development would cause delay.

Source: HP Journal August 1996 [JANDOUREK96].

Type [Serial Development \*364]: Process.

Related Concepts [Serial Development \*364]

• Evolutionary Project Management \*355

• Evo Step \*141

## Server Stakeholder \*651 May 21 2005, 2017

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

Related Concepts:

* Client Stakeholder \*650
* Stakeholder \*233

## Service: Concept \*526 February 24, 2002

(Provision of) a facility to meet the needs or for the use of a person or thing." <-OED

Providing the service is WHY a system does something.

Providing the service is WHY a system does a function

\* The function is WHAT the system does

\* A mechanism is HOW the system does a function ("means")

\* A process is (part of) HOW the mechanism does a function ("activity"). Don Mills NZ

Synonyms:

• Use, Uses?

## Set Concept \*133 May 3, 2003

A ‘set’ of things is a ‘list of terms’, that belong together in some defined sense.

Example:

My Boys: {Dag, Tor, Kai, Per}.

Or

My Boys: Set {Dag, Tor, Kai, Per}.

Keyed Icon [Set \*133]: { }

Type [Set \*133]: Parameter [Relationship], Collective Noun.

## Sibling Concept \*265 June 6, 2003

A *sibling* is a ‘parallel kid’ of any specification artifact such as a process, a function, a system or a delivery step.

‘Parallel kids’ are of the same level of derivation, that is, they have the same parent.

Example:

Ruth: Parent.

Kid [Boy]: Tommy.

Kid [Girl]: {Wendy, Linda}.

Kid [Boy] and Kid [Girl] (above example) are siblings (same Parent).

Wendy and Linda are siblings (same Parent).

Type [Sibling \*265]: Relationship.

## Side Effect Concept \*273 May 8, 2003

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’ ‘known or unknown’ and ‘negative, neutral or positive’.

‘Intended’ means that we have chosen the design because we knew about and valued those particular side effects.

‘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).

Related Concepts [Side Effect \*273]:

• Impacts \*334: This parameter is used to specify side effects.

Keyed Icon [Side Effect \*273]: \*.->

‘\*’ represents multiple impacts, including multiple side effects. ‘->’ represents the ‘Impacts’ symbol.

Example:

DesignX \*.-> {AttributeY, AttributeZ}.

Type [Side Effect \*273]: Design Concept, Scalar Metric.

## Sign "SIGN" : Concept \*195. August 16, 2001tg 1957

A ‘written sign’, containing the defined text.

Type: Condition Class.

Synonym for Indicator "indicator" . \*195

*Example:*

Red: Sign {any panel light constant or flashing red, any screen equivalent warning danger}

## Significant Specification [{words, text, diagrams}]. Concept \*134. August 16, 2001tg 2000

‘Significant’ refers to any specification content which, if wrong, misinterpreted, or missing, represents a potential for non-trivial downstream cost to correct and suffer.

*Note 1.*

*Such (non-trivial cost) specification defects are called Major defects.*

*Contrast with ‘Non-significant specification (comment) "comment" that contains potential for minor defects only.*

*Non-significant specification is typically found in commentary, notes, footnotes, introduction summary, and background. But ‘comment’ may be interspersed with significant specification.*

*Rationale: this distinction is useful so that we can focus our energy on significant specification in doing spec QC and in evaluating the density of Major defects.*

*Related Concepts:*

*• comment: the other kind of specification (non-significant).*

*• major defects: defects not found in non-significant specs*

*• SQC: a process which is guided strongly by this distinction, if focusses on the significant specification, rather than commentary.*

*• generic rules: there is usually one rule which insists on separation of significant specification and commentary.*

*Usage: Rules, Generic , insist on making a visual distinction between significant specs and non.*

Type: specification class.

Domain: engineering.SQC.analysis

## Softcrafter: Concept \*573 July 12, 2002

A Softcrafter is a person who practices the craft of programming software for computers.

This type of person is better known as a ‘programmer’ (or even ‘coder’). Sometimes they call themselves software engineers, without any engineering competence or qualifications. This is rather like a good carpenter calling himself a structural engineer or skyscraper architect.

Related Concepts [Softcrafter, \*573]

• Softect \*567

• Architect \*193

• Systect \*565

• Designer \*190

• software engineer \*571

• software \*570

• logicware \*575

## Softecture: Concept \*566. July 12, 2002, 2013 ‘concatenation’

Softecture is a concatenation of software and architecture. It is the highest level of design activity in a software engineering project, or in the software area of a systems engineering activity.

Related Concepts [Softecture, \*566]

• Softect \*567

• Architect \*193

• Systect \*565

• Designer \*190

Background: The terms softecture and softect were defined in my book Principles of Software Engineering Management 1988 [Gilb88]. As far as I know there is no other origin. They were coined by me.

## Softect: Concept \*567. July 12, 2002, Edited March 20 2013, edit apr 13 2013 ‘concatenation’

Softect is a concatenation of the words software and architect. A softect is one who does software or IT systems architecture.

Related Concepts [Softect, \*567]

• Softecture \*566 the activity performed by a softect.

• Architecture \*192 (noun), \*499 (process) , the generic process done by the specialist in software

• Systecture \*564 an activity that must co-ordinate and set constraints for the softect

• Systect \*565 a person who co-ordinates the softect, by considering a higher level of the system architecture and requirements .

• Designer \*190 a softect is a specialist type of designer.

Background: “The softect is a necessary function in a large software engineering environment, in which there are many specialist software engineers. The softect is the necessary synchronization and co-ordination function for the many specialized engineers and builders. The softect presumes that the software must be designed, and is only concerned with finding a technical solution set which will satisfy the multiple conflicting objectives of the user as well as possible. … The softect is also a specialist software engineer, the specialty being overall control of a complex engineering process.” [Gilb88, p. 258] The term ‘Infotect’ (\*568) was also coined to describe the information system architect in the same book.

Synonyns [Softect, \*567]

IT Architect, \*567

## Software Concept \*570 March 12, 2003

Software refers to the ‘non-hardware’ aspects or components of a system. It specifically includes computer programs, data (computer readable files and databases), and software documentation and plans (any form of specification or plans made by people concerning software).

Related Concepts [Software \*570]:

• Software Engineering \*572

• Specification \*137

• Documentation \*579

Type [Software \*570]: System Component.

## Software Engineer: Concept \*571. July 12, 2002

A software engineer is an engineer with specialty in software. They are characterized by the ability to assemble software components based on quantified attributes. This ability is aimed at the need to meet multiple quantified requirement performance levels, within specified resource constraints, and other constraint limitations.

Consequently software engineers think in terms of measurable system performance (including quality) characteristics, and costs for design, implementation, decommissioning, adaptation, and operation. They know how to access the multiple quantified attributes of a design component and how to measure these attributes in the systems they engineer.

Note: computer programmers (aka softcrafters \*573) are not engineers of any kind. But they commonly misuse the title. Fortunately such misuse is being outlawed in certain US states (Texas), and Canadian provinces (Vancouver).

Related Concepts [Software Engineer, \*571]

• Software \*570 all non hardware types for machines and people, such as computer programs (logicware), dataware ( like computer databases), infoware (like documentation and planware), and human-computer interfaces.

• Softcrafter \*573 a computer programmer, with no pretension of being an engineer.

• softect \*567. The softect is narrowly concerned with the *design* function of software engineering, while the software engineer has a *much broader territory* including non-design disciplines requirements, contracting, testing, and anything else necessary to making the real system work as planned finally and continuously for the system lifetime.

## Software Engineering Concept \*572 March 12, 2003, March 24 2010

Software engineering is the engineering discipline of enabling and motivating software systems to deliver a balanced set of values, directly or indirectly, to a balanced set stakeholders, throughout their lifecycle\*.

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.

The concept ‘balanced set of values’ (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. The inevitable constraints on the engineering solutions means that intelligent prioritization of how much, or which values, will be delivered to which stakeholders and when – must be intelligently considered, according to a defined policy for prioritization.

The concept ‘a balanced set of 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, owners, or customers, alone.

Related Concepts [Software Engineering \*572]:

• Software \*570

• Engineering \*224

• Systems Engineering \*223

• Stakeholder \*233

• Value \*269

Type [Software Engineering \*572]: Engineering Discipline.

• thanks for Ian Sommerville and Frans Ver Schoor for inspiring this revision

<http://se9book.wordpress.com/2010/03/23/semat-and-the-definition-of-software-engineering/>

## Solution "solution" : (Concept \*047) . July 16, 2002

A ‘solution’ is a design idea which, if implemented, is *expected* to lead to the partial or full satisfaction of a set of attribute requirements; to solve a (defined) ‘problem’.

Rationale:

A Neutral Term.

The word ‘design’ is more’ technical’, and the word ‘strategy’ is more from a management culture. So, when I need a more neutral, and simple, term to cover both uses, I use the term ‘solution’, meaning solution to a ’multiple requirement satisfaction problem’.

Related Concepts:

• A design specification \*586, by contrast to a ‘solution’, does not imply that anything is ‘solved’ (requirements met). A design specification can potentially have any effects, including counter-productive ones. A spec is not necessarily good.

• So a solution, by contrast to design specification, is a specified design idea which has been evaluated as being able to solve a defined requirements level problem. It is deemed to be a good design.

“His design idea was believed to be a solution to the design requirements.”

Keyed Icon [Solution, \*047]

: [Solution Name] and/or a rectangular box.

The underlining of the Solution Tag in the keyed icon is intentionally there to give a ‘bottom’ to the box, and to distinguish the ‘[\_\_\_]’ symbol from the simplified icon for ‘process’, ‘[Simplified Process]’ or a qualifier ‘[qualifier 1]’.

Synonyms: Design Idea and Strategy.

Type: problem solving concept

Domain: engineering.design.solution

## Source Concept \*135 May 9, 2003

‘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.

Synonyms [Source \*135]:

• Process Input Information.

Related Concepts [Source \*135]:

• Evidence \*063

Type [Source \*135]: Parameter, Document.

## Source (parameter) "SOURCE (parameter)" : Concept \*135. July 9, 2002

A source definition states where the written, or oral, source information is to be found.

*The source information answers the question:” Where can I find more detail on the definition, extract or conclusion?”*

*Example:*

Goal 60% <- SRA section 6.4.

SRA: Source: IEEE Annals of Semiconductor Research Abstracts.

Keyed Icon [Source \*135]: <-- (example Product <-Source).

Synonym: Sources.

Note:

There can be more than one source reference and this is generally handled by giving a list of sources in the Set parenthesis: For example:

Goal 66% <- {S1, S2, S3}.

Or

Fail [Euro] 60%. Source {S1, S2, S3}.

If you feel strongly writing the plural (Sources’), it is OK but not logically necessary. The source *is* the set!

Type: Planguage Parameter

Domain: engineering.specification.relationships,Source

## Space DELETE Concept \*380 February 25, 2003

Space is a specification concept that can include the geographic ‘Place’ notion, and in addition any other *notions of group or location*, such as component, market, religion, race etc.

Space refers to the set of possible scope dimensions that are neither temporal, nor event dependent, but which refer to ‘where’ notions.

‘Space’ is a parameter that we can use to define a set of dimensions for any purpose.

Space refers to any spatial notion such as:

• geography ( nation, city ) - this is same as ‘Place’.

• organizational (IBM, Nokia, US Government, UK MoD)

• people or thing types (novice, teenager, pensioner)

• system components (hardware, radio transmitter, software, database, user terminals, chips.

Space notions can all be identified in real three dimensions on a real system.

There can be any useful number of space dimensions in a single qualifier expression.

Space can be used as a parameter.

Space = {Buddist, Korean, Teenager}.

Related Concepts [Space \*380]:

• Time \*153

• Event \*062

• Qualifier \*124

• Scope \*419

• Place \*107

Type [Space \*380]: System Concept.

## Special Causes Concept \*019 November 21, 2002

Special causes are non-recurrent defect causes, and cannot be exploited by management to systematically improve a process organization-wide by means of changing the organization, process or system.

They are not caused by the organization (training, management, motivation, information, tools); for example, not by an incorrect work process standard.

They can be used to improve things more locally, at the level of the individual engineer. For example, by learning that they misinterpreted a work process standard, motivating, teaching or instructing the individual.

Note: Dr. Walter Shewhart (Deming and Juran’s teacher of SPC) called these “chance causes. The main point is that special causes may be treated by the individual worker, if they are given information and power, while common causes must be treated by management. W. Edward Deming's term (‘special’) is the preferred term in this book.

“A special cause of variation is something special, not part of the system of common causes. It is detected by a point that falls outside the control limits.” (DEMING93, pg. 179).

Synonym: Chance Cause.

Related Concepts [Special Cause \*109]:

• Common Causes \*020 (a related but different term)

• Statistical Process Control (SPC) \*466

• Systemic \*262

Type: Statistical Concept

Domain: Statistical Process Control, Continuous Process Improvement.

## Specific Concept \*136 January 11, 2003

Specific means applicable to a precise, definite, set of components. It is usually used to reduce scope to declare a component (or set of components) as being explicitly singled out as the subject for attention.

Example:

*Non-Euro Currency Display: Constraint [European Economic Community, After Next Year]: No Non-Euro currency denomination values allowed in shop windows <- Euro Commission Document AX-0013.*

A ‘*specific* constraint’ is a system specification constraint that applies to a specific set of defined conditions. *These conditions would typically be defined in [qualifier brackets].*

Related Concepts [Specific \*136]:

• Generic \*081

Type: Adjective, Scope.

## Specific Design: \*558 June 22, 2002

A specific design is a precise and definite design, intended to give the final range of performance and cost attributes, as a result of that design, to a real system.

It is distinct from all previous notions of design upstream, which are intended to narrow the choices in an appropriate direction; but not intended to make a final specific choice.

As usual, we would be wise to distinguish between the intended design, the actually implemented design, (which could be different from both the intended design and the specified design) and the design as specified (which could be defective in relation to the intent).

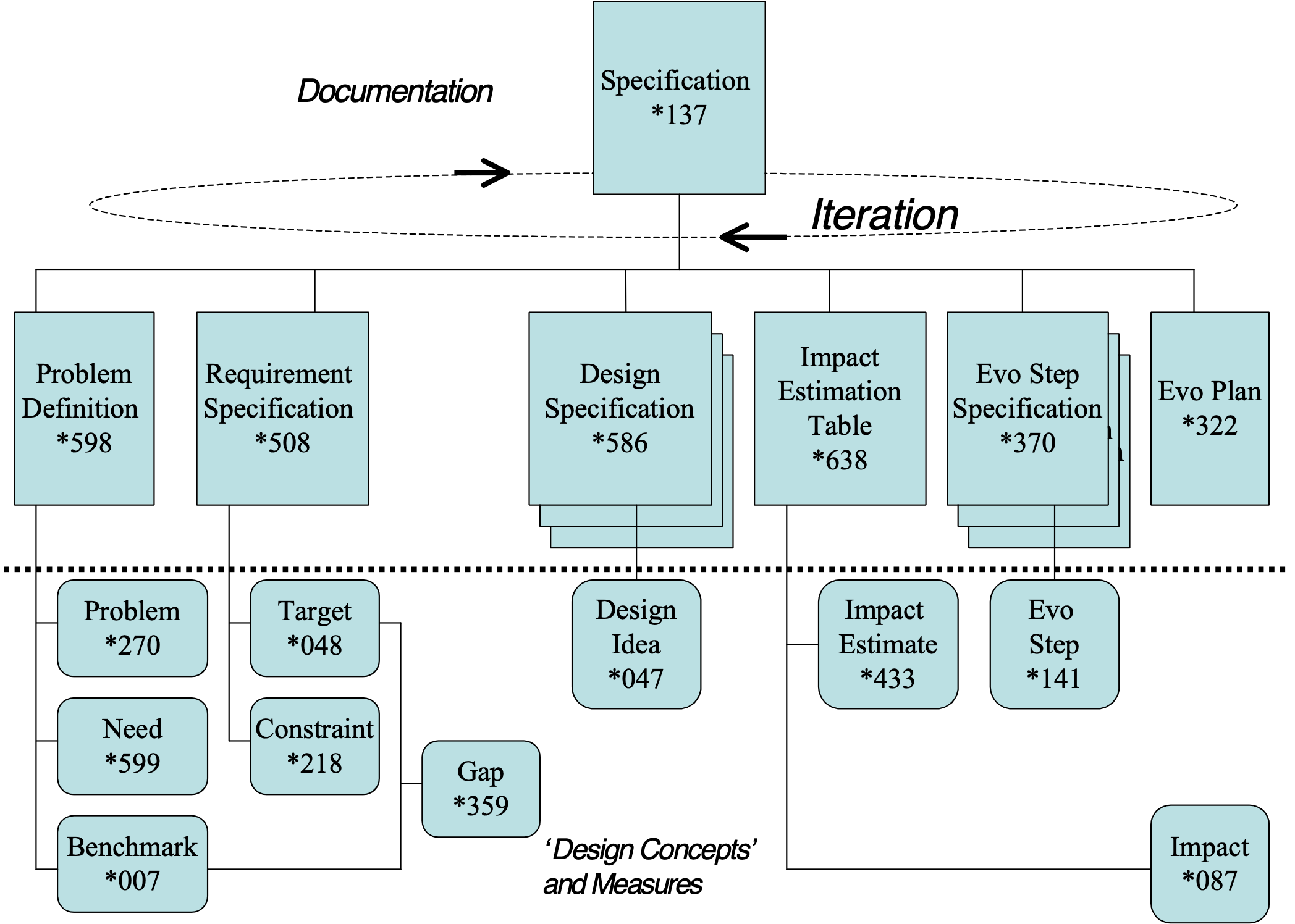
Related Concepts [Specific design \*558]

• function design \*521 a ‘less specific’ upstream design

• design \*047 any class of design

## Specification Concept \*137 November 12 2004, Aug 19 2010

A ‘specification’ communicates one or more system ideas and/or descriptions to an intended audience. A specification is a formal, written means for communicating information.

Figure \*137. Various types of specification.

Notes:

1. A specification is by definition written, but it could be temporarily oral with intent or capability of being written shortly thereafter.

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) and project plans (such as evolutionary delivery 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.

Origin:

SPECIFICATION (from Med. Lat. specificatlo, specificare, to enumerate or mention in detail), any detailed statement, especially one on which an estimate or plan is based, as the specification of a builder or architect. <-http://wordwizard.com/

Abbreviations [Specification \*137]:

• Spec: “Usually without a ‘.’. ‘Spec’ is often used as if it was a word, rather than an abbreviation.”

Related Concepts [Specification \*137]:

• Definition \*044

• Description \*416

• Documentation \*579

• Document \*180

• Planguage Concept \*188

• Commentary \*632

• Non-Commentary \*294

• Background \*507

• Core Specification \*633

• Planware \*577 ‘does not have to be written’

Drawn Icon [Specification\*137]: A standing rectangle. “The same as used for Document \*180.”

Type [Specification \*137]: Document.

## Specification Constant. Concept \*457. August 8, 2001 2250.

A specification constant is a Tag in a specification which always translates to a specific definition. The definition might be updated or improved, but it does not vary as a function of time or external environment.

Example [Specification Constants]:

Goal [Country =USA] 66%, [Newcomers] 50%

Newcomers: People who have never used our system.

But not this [A Specification Variable].

Meter [Customer Defined Acceptance test] <depends on current agreements> : Default: Acceptance Test Standard.

Related Concepts:

Scale Variable, \*446

Scale Qualifier, \*381

Specification Variable, \*456

Type: Planguage term type.

Domain: CE.Planguage.Specification.Term.Specification Constant.

## Specification Content Review (SCR) Concept \*542 November 21, 2002

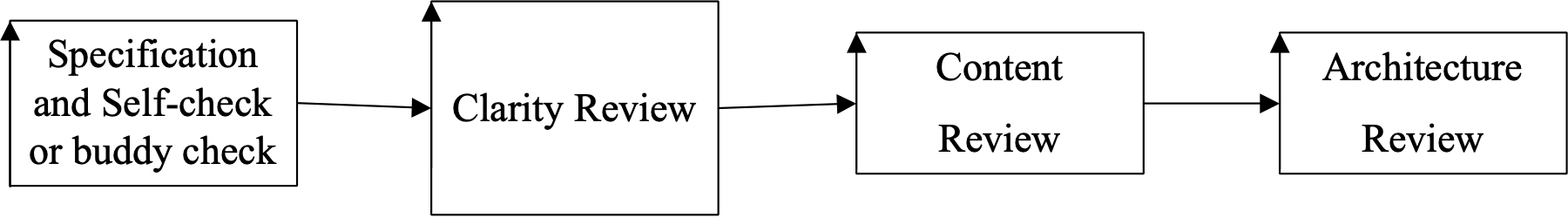
A class of specification review that focuses on the *goodness of the ideas* specified, no matter *how well* they are specified. It asks ‘is this the right stuff to do?’. It looks at *content of concept*, not format or packaging. It makes use of Content Rules to determine what to review in a particular type of content review.

Fig. \*542 There can be any number of different focus reviews on an engineering specification. There can be many potential methods for conducting a review, such as SQC (\*051) which hare very disciplined, objective, and quantified.

Note: it is imprudent to conduct a serious spec content review unless the specification has first successfully exited from a specification clarity review (\*607) – meaning the spec is considered well written, clear and complete – ready for content judgment to be passed by more senior people.

A specification content review can be run by checking that the applicable Content Rules (like Design Rules) are applied; or that their non-compliance is justified by some valid overriding consideration. A count of the density of violation of the applicable Content Rules can be a quantified basis for judging the specification in the content review; and for deciding exit from that review process (‘no more than 1.0 Major defects/page allowed’). An exit like this would say that the Content Rules have been reasonably followed. It would still not answer the even larger question like optimization within a larger set of specifications (‘do we have the best design?’)

Abbreviation: SCR

Synonyms SCR \*542:

• Specification Concept Review \*542

• Content Review \*542

Variations SCR \*542: (examples among many possibilities)

• Design Concept Review

• Requirement Concept Review

• Architecture Concept Review

• Test Plan Concept Review

• Value Review: aimed at understanding the value of a specification, the knock on effect of the content

Related Concepts [Specification Concept Review, \*542]

• Content Rules \*543

• Review Criteria \*541

• Spec QC \*051

• Specification Intelligibility Review (SIR): Concept \*607

• Design Rules \*593

• Content \*594

• Review Focus \*608

Type: systems engineering, review process

Domain: management analysis of ideas

## Specification Defect Concept \*043 March 23 2006 (Threat added)

A specification defect is a defect in a specification resulting from a failure to observe a formal, written, required rule about how the specification should be written. More simple: **defects are rule violations.**

A specification defect can be classified as either a major defect (Threat) or a minor defect depending on the estimated cost of fixing the defect (including any resulting problems) at a *later* stage in the system lifecycle.

Notes:

1. In common usage a defect and a fault are not distinguished from each other. Here we want to use the terms so that ‘specification defect’ refers to a specification problem, and fault refers to a real artifact problem with actual potential for a malfunction to occur as a result.

Error/Slip 🡪 (Specification Issue) 🡪 Specification Defect 🡪 Fault/Bug/System Defect 🡪 Malfunction

2. Further, we are sharply limiting the notion of ‘specification defect’ to a violation of some agreed and defined standard for specification (a specification rule), not just one person’s personal opinion about ‘style’.

Synonyms [Specification Defect \*043]:

• Defect \*043

Related Concepts [Specification Defect \*043]:

• Major Defect \*091

• Minor Defect \*096

• Error \*274

• Specification Issue \*529

• Fault \*339

• Malfunction \* 275

• Specification Rule \*129

Type [Specification Defect \*043]: Risk Analysis

## Specification Element Concept \*506 May 24, 2003

A specification element is any distinct element, however elementary or complex, of a specification.

A specification element can be a statement, an expression or a term.

Related Concepts [Specification Element \*506]:

• Specification \*137

• Statement \*140

• Expression \*302

• Term \*151

Type [Specification Element \*506]: Specification.

## Specification Intelligibility Review (SIR): Concept \*607 September 3, 2002

See Clarity Review \*607

## Specification Issue Concept \*529 January 13, 2003

A specification issue is a checker-*perceived* potential defect in any form of documentation or system development specifications.

Specification issues, when resolved by the author or editor either become acknowledged defects or not.

Notes:

1. A specification issue is a polite way for a Specification Quality Control checker to say, “I think I may have identified a defect, but I choose not to say for sure just yet, and leave the decision to the author.”

**Error/Slip -> (Specification Issue) -> Specification Defect -> Fault/Bug/System Defect -> Malfunction**

2. A specification issue can be directly inserted into the specification using the parameter ‘Issue’. When it is resolved it can either be removed, or the resolution can be documented.

Example:

Fault Level:

Ambition: Extreme low fault level.

ISSA: **Issue**: ‘ Neither <extreme> nor <fault level> used in the Ambition statement are well defined and agreed.

**Resolution** [ISSA]: They are now defined below <- TG Feb 8.

Fault Level: Scale: Bugs per month experienced by individual users (even if not reported to us).

Extreme: Goal: 0.1 bugs/month/user.

Synonym [Specification Issue \*529]: Issue. “A short informal form used in SQC context.” Actually an abbreviation.

Related Concepts [Specification Issue \*529]:

• Issue \*276

• Specification Rule \*129

• Error \*274

• Specification Defect \*043

• Fault \*339

• Malfunction \*275

• Resolution \*525

Type [Specification Issue \*529]: Risk Analysis.

## Specification Language: Concept \*248. March 4, 2003

Specification Language provides the structures and formats to express Planguage specifications.

For example Planguage parameters, icons, concept definitions and specification conventions are the formal skeleton onto which the product language is integrated.

Related Concepts [Specification Language \*248]:

• User Defined Term \*530

• Process language \*246. A project’s Specification Language must also be distinguished from its ‘Process Language,’ which is the ‘process standards for doing the project.’

• Project Language \*247: The set of all Process Language, Product Language and Specification Language artifacts used by a defined project. Within Project Language, a variant of Basic Planguage’s Specification Language can be used to express specific project specifications.

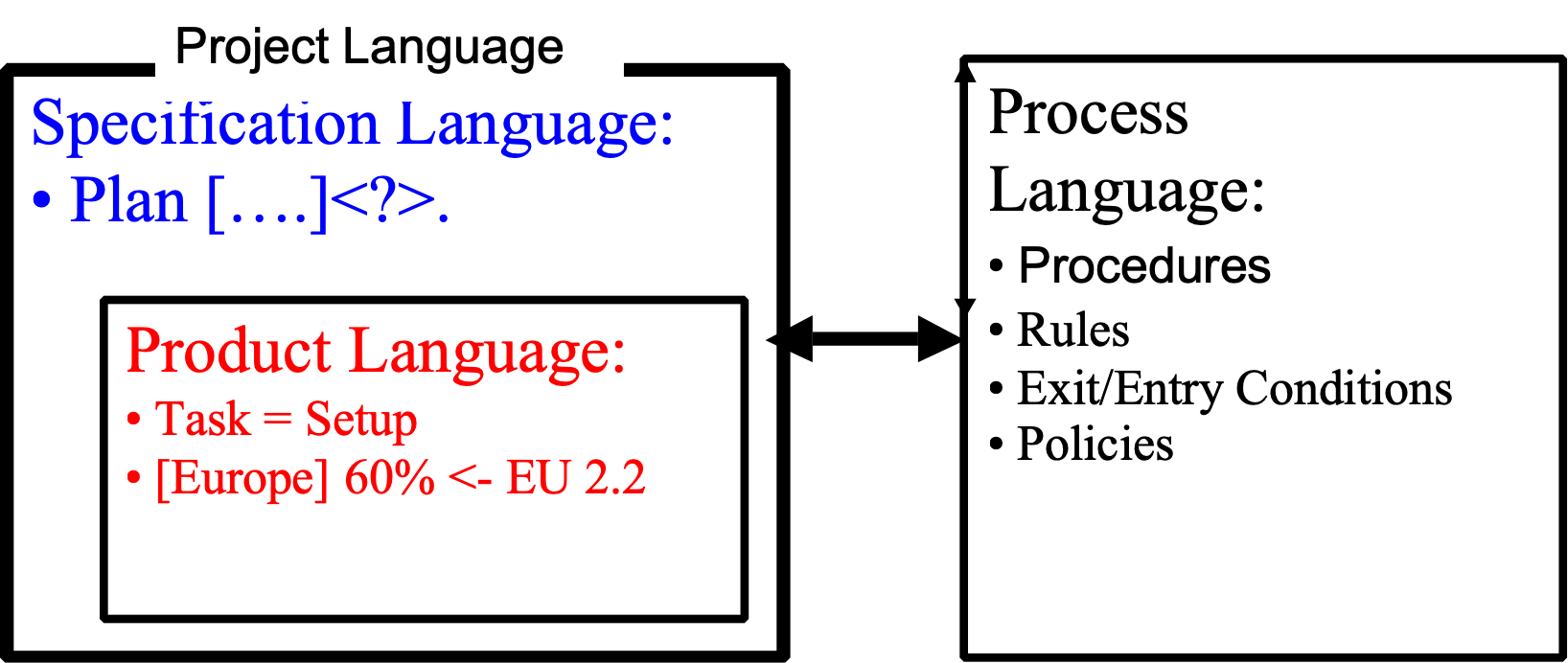
System Description Language (SDL) \*183. The subset of Planguage for describing systems, including ‘processes viewed as systems with attributes’.

Synonym: Spec Language, .

Type: Planguage class.

Domain: engineering.specification.Planguage.Specification Language

Rationale [Spec Language]:I felt a need to distinguish amongst different aspects of Planguage. I needed to give the different aspects of Planguage a formal defined name. This concept is of interest to people who develop specification standards. It would not be used by people who write them.



## Specification Maturity. Concept \*555. March 24, 2003

Specification Maturity is the level of evolution of a specification, standard or process artifact.

Specification Maturity is a parameter used to describe the level of evolution of a specification.

Specification Maturity classification does not express the specification quality level, which is a different concurrent view of a specification.

Related Concepts [Specification Maturity \*555]:

• Quality Status [Specification] \*463

• Quality Level [Specification]\*360

• Status [System] \*174

Example:

Specification Maturity [June 19 2002] Under Design <- TG

Suggested Specification Maturity Values (from Leo Hippeläinen, Nokia, June 19, applied on his projects)

Captured - The feature (or requirement) is identified.

Analysed - The dependencies and impact of the feature are analyzed. Work effort can be estimated.

Scheduled - The work for the feature has been entered to project timetables.

Under Design - component design work that includes this feature has started.

Under Implementation - component implementation work that includes this feature has started.

Under Component Test - component testing work that includes this feature has started.

Fit for Integration - component that includes this feature is ready for integration tests.

Available - Software release that includes this feature has been published.

Suggested by [leo.hippelainen@nokia.com](mailto:leo.hippelainen@nokia.com) June 18-19 2002.

Type [Specification Maturity \*555]: classification concept, parameter

## Specification Meeting Concept \*120 November 20, 2002

An SQC sub-process. A meeting of checkers and team leader. Scheduled after the individual Checking sub-process. The overall purpose is to evaluate the results from the individual checking activity.

A Specification Meeting sub-process has a variety of purposes including initial reporting of individual checking results and analysis, reporting information for the specification owner, such as issues, process improvement suggestions, and ‘questions of intent’ to the author/writer and additional checking. An Editor Advice Log is produced as written output.

A specification meeting also serves to motivate, train, stimulate, inform and to avoid duplication of reporting. It can be used to continue to find major issues in the documents, in addition to those discovered in individual checking by checkers. It has a speed and flavor of ‘brainstorming’ rather than finality.

The meeting duration and content are determined by a quick initial analysis of checking data from individual checkers, reported at the beginning of the meeting. At one extreme the checking data analysis can establish that there is no point in continuing the specification meeting.

Synonyms: {Logging Meeting, Product Meeting, Specification Meeting, Inspection Meeting [GILB93]}.

Type: Process.

Domain: SQC.

## Specification Quality Control Concept \*051 May 26, 2003

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 [GILB93]. 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.

Figure \*051: Sequence of SQC processes leading to review(s)

• 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]

Acronym [Specification Quality Control \*051]:

• SQC \*051

Synonyms [Specification Quality Control \*051]:

• Inspection \*051

• Peer Review \*051

“For additional specialized synonyms, see [WHEELER96].”

Related Concepts [Specification Quality Control \*051]:

• Quality Control \*279

• Specification Defect \*043

• Specification Rule \*129

• Specification Review Rule \*543

• Review \*197

Type [Specification Quality Control \*051]: Method or Process.

## Specification Quality Level Concept \*360 May 9 2005

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 [SpecificationQuality Level \*360]:

• Major Defect Density \*360

• Quality Level \*360

Related Concepts [SpecificationQuality Level \*360]:

• Status \*174

• Remaining Major Defects/Page \*060

• Specification Defect \*043

• Major Defect \*091

• Logical Page \*103

• Specification Maturity \*555

Type [SpecificationQuality Level \*360]: Parameter [Specification Control].

## Specification Reference. Concept \*458 August 8, 2001 2311

A specification reference is any user-defined Tag which cross references the defined term elsewhere.

Note 1. This does not include Planguage parameters.

Related Concepts:

• Specification Variable \*456

• Scale Variable, \*446

• Scale Qualifier, \*381

• Specification Constant, \*457

•

Type: Planguage specification type.

Domain: Planguage.specification.Tag reference.

## Specification Review Rule Concept \*543 May 25, 2003

Specification review rules specify the official known best practices for deriving a systems engineering concept (for example, a design) from other information (like requirements and known technology).

A specification review rule is used both to instruct the engineer in how to carry out a systems engineering process, and then consequently to instruct reviewers on how to evaluate that resulting specification.

A specification review rule is a type of standard.

Examples:

Rules.SRR:

Gist: Some examples of Specification Review Rules.

SRR1: All critical qualities for defined stakeholders must be specified at performance levels validated as valued by stakeholders themselves (Architecture Content).

SRR2: The design should clearly the best one we can afford, risks considered, within our budget constraints? (Design Content).

SRR3: The design should be competitive, at least 2 years after latest product release, against estimated competitive Trend benchmarks (Design Content).

SRR4: The full range of risks must be ruthlessly documented, and the risks must be acceptable (Both Design and Architecture Content)

Notes:

1. Specification review rules inform us about ‘what’ should be specified. There is the notion of whether ‘what is specified’ is the ‘right thing’ for the organization or system concerned.

Related Concepts [Specification Review Rule \*543]:

• Specification Quality Control (SQC) \*051

• Review \*197

• Specification Rule \*129

• Standard \*138

Figure \*543: Specification rules and Specification review rules being used in SQC processes to decide if the content of a systems engineering specification is up to the best practice standards

Type [Specification Review Rule \*543]: Specification Standard.

## Specification Rule Concept \*129 May 26, 2003

A specification rule is a type of standard for specification.

Specification rules focus on ‘Is the specification well written?’ A set of specification rules states how we should both ensure and check, such things as the completeness, the unambiguousness, the accuracy, and the intelligibility of the information - in a specific type of specification.

In particular, a specification rule does not go deeply into an engineering process in the way that a specification review rule does.

Example:

Rule 2: Quantify Quality: All system quality requirements must be specified using a defined scale of measure and at least one target level of requirement. M

The ‘M’ is a code indicating that a rule violation is pre-classified as severity ‘Major’ defect.

Notes:

1. In an SQC process, a specification rule can be used to judge the specification quality (‘defect-freeness’ according to current rules) of any written output, from any specification process, which is subject to the rule. Violations of rules are defined as ‘defects.’ Rules give us an objective basis for judging conformance to the specification standards.

2. Specification rules determine ‘how well written’ a specification is. They do not pretend to judge ‘how significant or correct’ the contents of a specification are. (That must be judged by other means, such as further SQC (using specification review rules) and comparisons to external information. The more clearly written, and detailed, a specification is, the better chance we have to discover *content* weaknesses, and other inaccuracies in the specification.)

Related Concepts [Specification Rule \*129]:

• Standard \*138

• Specification Quality Control (SQC) \*051

• Specification Review Rule \*543

Type [Specification Rule \*129]: Specification Standard.

## Specification Type: Concept \*398. August 17, 2001 tg 1711

A spec type is the declared or inherited (from supra level specification) class of specification. *For example: Type: Quality Requirement*.

Usage 1. It can be declared explicitly (Type: ), after the tag. It can be implicit (Tag: <Type>: <the spec>. ). See the example below for instances.

Usage 2. A spec type can be declared at a higher level of specification. For example:

*Sub-Component X: Type: Requirements:*

*Reliability: Quality Requirement: Ambition…, Scale…, etc.*

*Navigation: Function Requirement: the astronauts must be able to navigate manually.*

*Eavesdropping: Design Constraint: eavesdropping must be enabled if local law dictates it.*

*Budget: Cost Requirement: Scale: Engineering Hours, Goal [To Release 1] 10,000* 🡸 *Boss.*

*Sub-Component Y: Type: Design. “Supra-level: all below must be some class of design”*

*D1: Component: ……..*

*D2: Process: …….*

*D3: Architecture: ………*

Usage 3. A ‘Spec Type’ can be given with any Planguage glossary-defined term, and in addition with any term defined by you locally.

## Specification Variable Concept \*456 May 1, 2003

A specification variable is any term in a specification which references another term, which we must see the definition of, in order to interpret it.

Example:

Design Idea S: Use [Latest Version] of the Server Software.

Latest Version: Defined As: The most recent version released in the country of use and available to the customer, whether they used it or not.

Where ‘Latest Version’ has to be assigned the number of the latest released version.

Synonyms [Specification Variable \*456]:

• Variable \*456

Related Concepts [Specification Variable \*456]:

• Scale Variable \*446

• Scale Qualifier \*381

Type [Specification Variable \*456]: Specification Concept.

## Specification Writer: Concept \*004. August 17, 2001 tg 1834

Synonym: Author, spec writer, cognizant engineer.

## Specify, To Concept \*239 May 9, 2003B

To ‘specify’ means to articulate an idea.

Notes:

1. We can specify in writing or orally, with text and/or symbols. Planguage is written specification that can be driven by oral specification, (for example, capturing stakeholder needs by interview (oral specification), or as an input for formal requirement specification (written specification).

Synonyms [Specify, To \*239]:

• Spec \*239: As in ‘to spec.’

Type [Specify, To \*239]: Verb.

## Sponsor Concept \*396 May 6, 2003tg

A sponsor is a person or group, who has an interest in supporting the achievement of specific system change.

Notes:

1. Sponsors contribute, or are willing to contribute, resources towards the achievement of system change. The resource can be of a variety of types, such as financial budget, people, materials, reputation, or vocal support. The support can be limited to political or managerial power. They are one type of stakeholder.

2. Non-sponsor stakeholders have some interest in the system outcome, but they do not inject *resources* into the activity.

3. System change can have more than one sponsor.

4. Identification of the sponsors in a specification is useful, in order to find out whom to go to, in order to get continued or increased levels of support/resource. It sends a signal to specification readers that these are the people they are dealing with, and should aim to please.

Efficiency:

PP: Sponsor: Production Planning <- Email 16th April Jenny Jones to Alan Aldern.

Scale: Work Units produced per Work Day per Employee.

Goal [Next Year]: 3,000 Work Units/Work Day/Employee.

Stretch [In 3 Years Time]: 4,000 Work Units/Work Day/Employee.

CEWD: Sponsor [Stretch]: Corporate Enterprise Workforce Development.

Example of a specification of sponsor integrated with a performance requirement. Notice the two levels of sponsorship: one overall (PP) and one for the Stretch goal (CEWD).

Related Concepts [Sponsor \*396]:

• Stakeholder \*233

Type [Sponsor \*396]: Parameter, Role [Stakeholder].

## SQC: Concept \*051. August 17, 2001 tg 1909

Abbreviation for ‘Specification Quality Control’.

## Stakeholder Concept \*233 July 3 2017 Origin. Note 3

**A stakeholder is any person, group or object, which has some direct or indirect interest in a *defined* 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.

Stakeholder :

An interested party having a right, share or claim in the system or in its possession of qualities that meet their needs.

Source: Standard ISO/IEC 15288:2002] and ISO/IEC 25000

NOTE Stakeholders include, but are not limited to, end users, end user organisations, supporters, developers, producers, trainers, maintainers, disposers, acquirer, supplier organisations and regulatory bodies

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

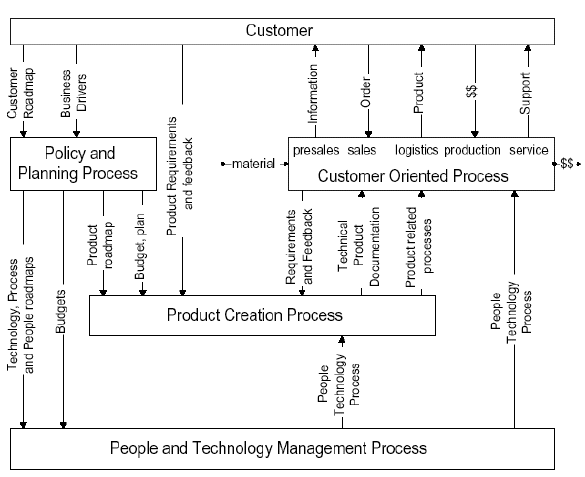
• using the product or system, even indirectly

• 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.



Some stakeholder concepts. Courtesy Gerrit Muller, Philips, Eindhoven, NL. In

“Requirements Capturing by the System Architect”[[14]](#footnote-14)

[Editor Note to Publisher: EDIT NOTE: PERMISSION GRANTED TO USE THIS IN EMAIL 9 NOV 2001 FILED IN PERMISSIONS FOLDER TG]

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!

4. Origin (added July 3 2017) "The term “stakeholder” has been widely used in management literature, and may be traced, for management popularization, to Freeman’s seminal book, “Strategic Management: A Stakeholder Approach” (1984). Freeman defined stakeholders as any identifiable group of individuals, who can affect the achievement of an organization’s objectives, or be affected by the organization’s objectives. Some of the key ideas in the stakeholder framework are that the management needs to specifically recognize and cater to the needs of all stakeholders, going beyond the traditional boundaries of investor, customer, vendor and employee. In many contexts, the term is also associated with corporate responsibility, since such a definition of stakeholders is inclusive of the public at large, civil society organizations and the Government.

Stakeholders include not only financial claimants, but also employees, customers, communities, governmental officials, ("and, under some interpretations, the environment, terrorists, blackmailers, and thieves.") (Jensen, 2000, p. 1)." SOURCE Stakeholder Analysis of IT Applications for Microfinance, BY NV KRISHNA, [nvk2005@yahoo.co.in](mailto:nvk2005@yahoo.co.in)

<https://www.linkedin.com/in/krishna-nyapati-696a772/?ppe=1>

AND PAPER AVAILABLE AT http://www.gilb.com/dl446

Example:

Goal [Stakeholders = {Stakeholders = Installers, Service People}, Deadline = 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.

5. Stakeholders can be internal or external to a system. Internal stakeholders are typically in our development organization, or related to it or to the development process. External stakeholders are users and customers of the developed system. 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.

6. Stakeholder classes RACI, which stands for Responsible, Approver, Consulted, and Informed. This might provide valuable detail rather than a simple clump of people within the stakeholder. <- Erik Simmons

7. : I suggest we can generally classify stakeholders with these concepts:

Stakeholder Hierarchy

= any set of Client Stakeholders and their Server Stakeholder relationships.

Client Stakeholder

= any stakeholder with needs that any server stakeholder might satisfy.

Server Stakeholder

= any stakeholder that plans to satisfy some client stakeholder needs.

Keyed Icons [Stakeholder \*233]:

: - | Stakeholder or Neutral Stakeholder converts in Word to 😐

: - ) Happy/Satisfied Stakeholder converts in Word to ☺

: - ( Unhappy Stakeholder. Converts in Word to ☹

O<- Requirement (O) source <- (experimental idea June 2017) or ◦ **<-O**

Related Concepts [Stakeholder \*233]:

• Owner \*102

• Client \*235: Synonym is Customer

• Sponsor \*396

• Consumer \*038: Synonym is Customer

• Decision-Maker \*237

• User \*234

* Designer \*190
* Requirement Generators \*233 (2017)
* Requirement Sources \*233 (2017)

Types [Stakeholder \*233]: Parameter [Role], Role.

## Standard Concept \*138 January 2, 2004

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*[[15]](#footnote-15)*

Standards include: {rule, policy, process, entry condition, procedure, exit condition, form, template}.

Synonyms [Standard \*138]:

• Work Process Standard \*138

Related Concepts [Standard \*138]:

• Rule \*129

• Policy \*111

• Process \*113

• Form \*068

• Template \*254

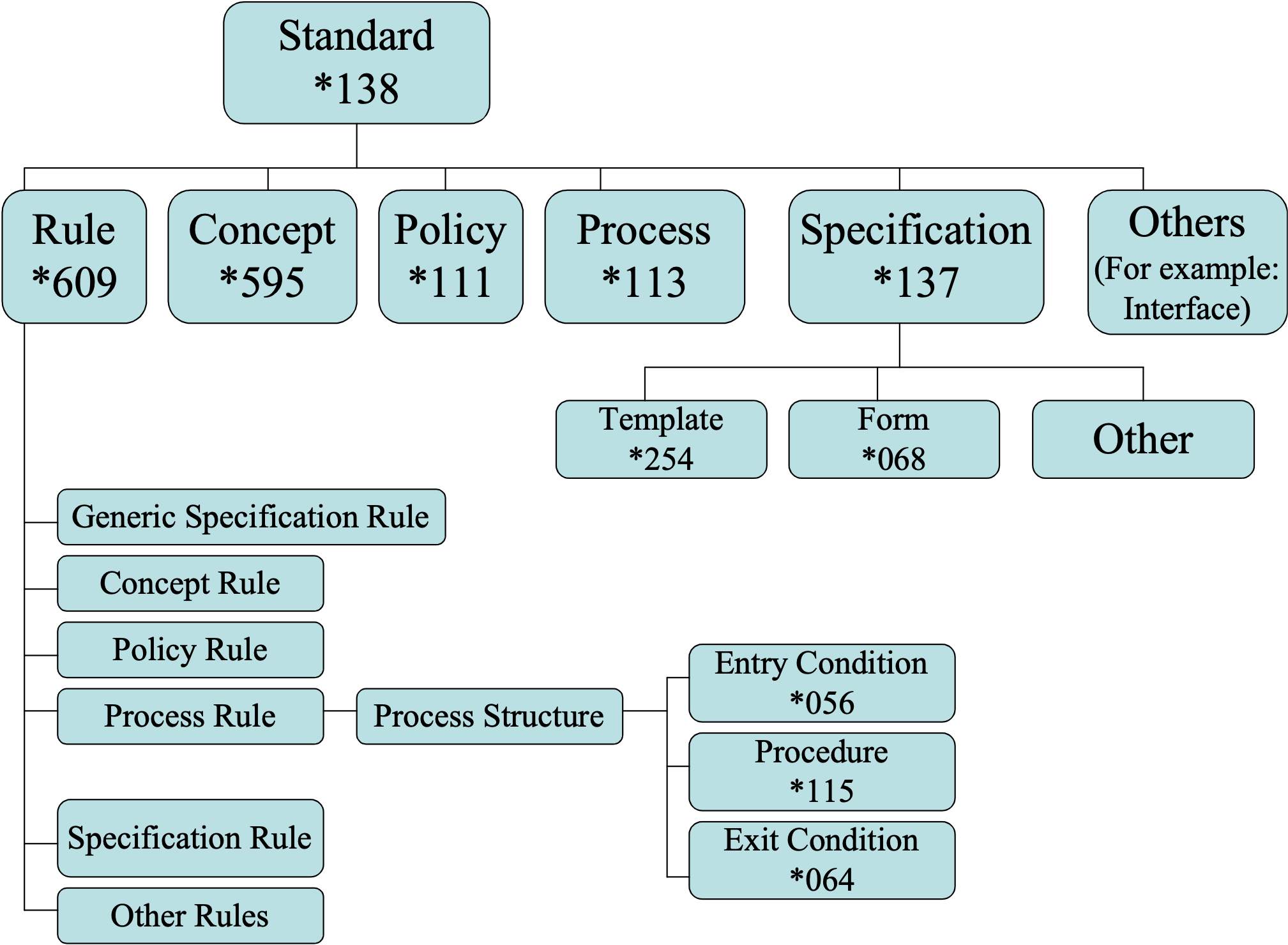


Fig. \*138 There are a variety of work process standards provided by Planguage to help define work processes.

Type [Standards \*138]: Systems Engineering Concept.

## Starts: Concept \*315. August 17, 2001 tg 1918

Indicates a condition where a defined event begins.

Type: Planguage event relationship parameter

Domain: engineering.specification.processes.relationships.Starts

Note 1. Start says nothing about duration, whether it is still in process, or completion of the event.

Example:

X [ A Before [B Starts] ]….

Y [ B After [C Starts] ]…..

Meaning: X is true if A is true before B (as a time event) begins.

So, if A only existed at or after the beginning of B then X would not be valid.

Similarly Y is valid only if B is true after even C starts.

Peace [ Rebels Cease Fire Before Government Troops]

Goal [Peace] Sign Treaty.

Peace is true, and then we plan to sign a treaty,

if the rebels cease fire before the government troops cease fire.

Examples (using icons) :

X [A• => B\*] “A Ends (•) before B Starts”

meaning X is conditional upon A being (or completed) and thus true (A•) before (=>) B starts (B\*),

while . The ‘•’ means that A must be brought to completion before B can start. This is different from A^ (meaning A is in process and has not stopped yet).

X [A\* => B\*] informally X[A => B] is the same but not as precise.

X [A^ => B•] A is still in process before B is stopped

meaning X is conditional upon A being started before B starts , A does not have to be completed or true before B starts.

Related Concepts:

• Ends \*316 Icon •

• Before \*312 Icon =>

• After \*313 Icon =>

• In Process \*113 ^

Keyed Icon [Starts, \*315]: ‘\*’

(an explosion symbol and contrasted with ‘•’ = ends \*316).

## State: Concept \*174 February 16, 2003

A state is a defined status of a specified system or component.

A state is a set of circumstances or attributes characterizing a person or thing at a given time; way or form of being; the condition, the status. <- Dictionary definition.

States can vary in operation, or during projects.

A State parameter documents a basic specification qualifier for the planning or design.

Example:

Goal [State [Device-M] = Off ]: 33%, [State [Device-M] = On]: 95%, [State [Device-M] =Undetermined]: 0%.

Note: Off, On and Undetermined are Planguage predefined States. Other states (example Black, Blue, Hot, and Cold) may be defined in addition to predefined states.

This example means that the Goal level (on a Scale defined above it, but not shown here) is 33% when the state of Device-M is Off. The Goal level when the device is On is 95%.

Keyed Icon [State \*174]: [!]

A condition ‘[?]’ that has been determined ‘!’.

Synonym [State \*174]:

• Status \*174

Related Concepts [State \*174]:

PREDEFINED PLANGUAGE STATES:

• On \*625

• Off \*626

• Undetermined \*627

• Undeterminable \*628

• Unknown \*629

• Unstable \*630

• Changing \*631

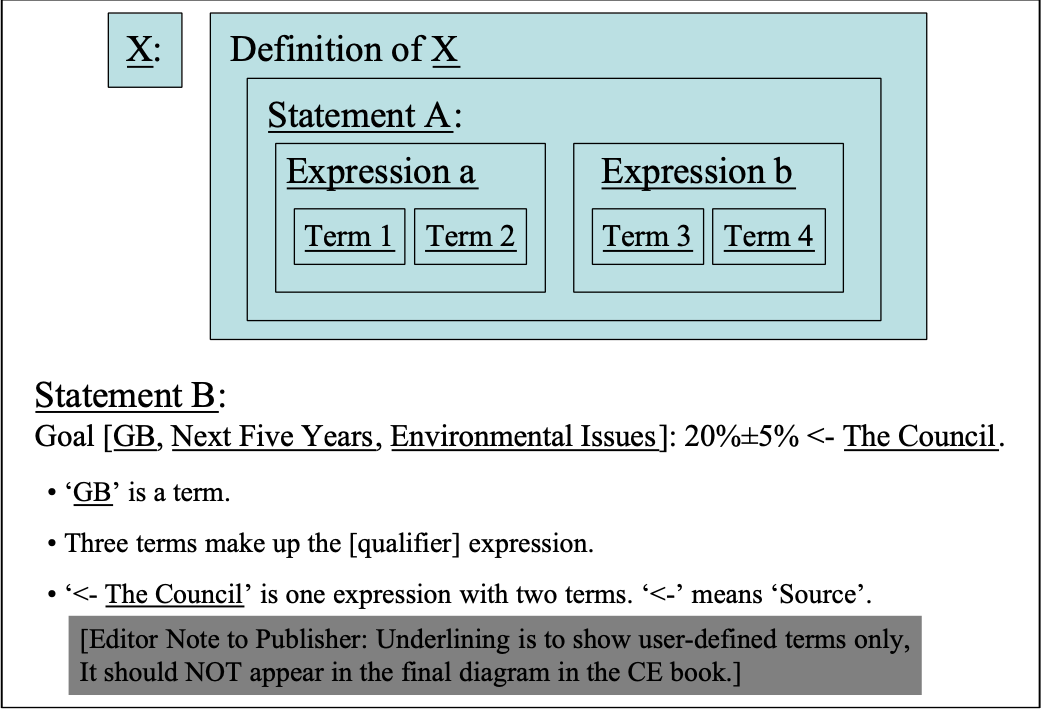
• True \*625

• False \*626

Type [State \*174]: Parameter, Condition Concept

## Statement Concept \*140 February 25, 2003

A Planguage statement is one or more Planguage expressions connected to form a meaningful and complete specification.



Notes:

1. Numerous different classes of Planguage statement exist, including {Requirement, Design, Plan, Relationship, Note, Parameter, Condition, Function}.

2. A statement would *normally* begins on a new line, and ends with a ‘**.**’.

Related Concepts [Statement [Planguage] \*140]:

• Term \*151

• Expression \*302

• Definition \*044

• Specification \*137: A ‘specification’ communicates one or more system ideas and/or descriptions to an intended audience. A specification can consist of many statements.

Type [Statement [Planguage] \*140]: Specification Concept.

## Statistical Process Control Concept \*466 January 21, 2003 B

A method for analysis of statistical process data, which aims to improve the performance levels and reduce the variability of the results. Emphasis is placed on distinguishing between systemic problems (which can be cured by changing the system) and special causes of results (which need to be improved at the individual, not the organizational) level.

Source: Developed by Walter Shewhart and taught by his students W. Edwards Deming and Joseph Juran. See references to these authors.

Acronym [Statistical Process Control \*466]:

• SPC \*466

Related Concepts [Statistical Process Control \*466]:

• Plan Do Study Act Cycle (PDSA Cycle) \*168

• Continuous Process Improvement (CPI) \*424

Type [Statistical Process Control \*466]: Method.

Domain [Statistical Process Control \*466]: CPI.

## Status Concept \*174 April 28, 2003B tg

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".

All five lines of examples are examining different types of Conditions Status is used to collect the results of the condition set tagged {A, B, C & D}. 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, 20xx. 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 [Status \*174, Specification Status]:

• 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 [Status \*174]:

• State \*174

Related Concepts [Status\*174]:

• Condition \*024

Type [Status \*174]: Parameter [Specification Control, Process Control]

## Step Element Concept \*409. January 3, 2003

A step element is any component of a step specification.

In practice this can be one or more of the following:

• a design idea (including strategies, architecture, solutions etc.)

• a function requirement

• a step specification defined elsewhere, but reused in a new qualifier context, such as Step-ABC [Utah] meaning Step-ABC applied to the state of Utah.

• a step task

Related Concept: [Step Element \*409]

• Step Task \*347

Synonym: Step Component.

Type: Specification Concept

## Step Level Concept \*535 January 3, 2003

A Step Level specification is a sub-class of a Goal level. A Step level is intended for specification of intermediate scalar levels of both performance and resources, where these levels are not primary product or system targets in themselves. They are just approximate plans for evolutionary steps or other increments of project or system maintenance plans.

Usage:

The step levels can be used instead of a Goal parameter, in order to highlight and emphasize that these are just planned levels for intermediary stages of system development. They are used to motivate engineers to make a degree of progress, and to give us feedback when we do not. But they emphasize that these are just intermediary milestones and not primary or final targets for the system.

Related Concepts:

• Goal

• Path Plan

Keyed Icon [Step Level \*535]: -v-

Type: Target.

NOTE: THIS IS AN EXPERIMENTAL DEFINITION WITH NO PRACTICE BEHIND IT. IT WILL NOT BE INCLUDED IN THE CE BOOK. TG JAN 3 2003

## Step Task Concept \*347 January 3, 2003

A step task is any component of an Evo delivery step that has to be carried out in order to deliver the planned impact, but is not itself a specific design idea or function (step content) to be delivered to the host system.

Example: template of normal Evo step tasks. Normal Evo Cycle Process:

Entry Process {Conditions: previous step exited}

Plan: (design)

Spec: Cycle Requirements? Measurable, testable   
( what we will deliver at end of Evo cycle)

Confer/agree with impacted stakeholder on the goal level and type.

Spec: Step “Designs” (if any) to implement (impact Quality & Cost)

Spec: Step “Functions” (if any) to implement

Plan: Estimate: Impacts on Quality and Impacts on Costs.

Reduce step content, if we exceed “2%’ step size guidelines.

Plan: tasks for the week, allocate to people

Do: implement the tasks according to plan

Acquire suitable hardware/software platform

Acquire suitable people {Stakeholders and Users}

Build/Acquire/Code

Test (Internal): should we bother to hand over to a stakeholder?

Install/deliver with Stakeholder using suitable users (Micro Field Trial, Practical Real World test)

Who manages this micro field trial? Let us say the test group.

Study:measure the results (quality against the required plan, costs, side effects)

Act: decide what to do about this ‘feedback’ in relation to our plans

Re-do it properly

Select next step (based on our experience and possible new signals)

Update project overview information and communicate to ‘Management’

Exit process {Conditions: met our targets, no unexpected problems)

Note 1. [Step Task content] Step tasks are things like planning, measuring, getting approval, training, quality control and testing which must be planned and done during the delivery cycle.

They are necessary overheads for moving the design ideas and function into the host system. They are like the activities of the cook, purchasing, measuring, heating, mixing, tasting, and serving the food to be delivered in a ‘cooking cycle’.

Step tasks are the process components of a delivery cycle.

Related Concept [Step Task \*347]

• Step Element \*409

Type: Evo planning component

## Story: Stakeholder Story § Concept \*665 January 6 2011, January 15 2019

A **story** is a very high-level definition of a **requirement**, containing just enough information so that:

1. It can be made more-structured, and better clarified, as a proper formal requirement, if necessary, for more specific purposes. Like for more accurate estimations of values and costs.

2. It has enough information to enable a rough understanding of value expected, stakeholders involved, and costs involved; as opposed to having little clear and sufficient understanding at all.

3. It will, as a minimum, contain

a. Some notion of at least one interested stakeholder

b. Some notion of a requirement of any type (Value, Function, Design, Constraint etc.)

c. Some notion of ‘Why?’. Justification, purpose, expectation, knock-on (higher level) result.

Keyed Icon: \*665 § (the paragraph sign, also known as “Section Sign”. Here it is chosen because it represents a ‘new thought '(Story) in a specification text (that is the old tradition of the typographical use of the symbol)., and SS § = **S**takeholder **S**tory

<https://fsymbols.com/computer/paragraph/>

The following types of stories are classified in Planguage, and for implementation in the needsandmeans.com/Valplan.net tool as of 15. 1.2019.tbd

1. **User Stories**: identical with the popular Scrum, agile methods from Mike Cohn
2. **Stakeholder Stories**: to broaden the user concept to include any class of stakeholder-
3. **Value Storie**s: where some quantified variable notion of what people want is specified, even if poorly (Extremely high security, cost reduction.)
4. **Solution Story**: a specific technology or solution (Design Idea \*047) is desired, in the hope of getting some result at a higher level. Similar to ‘Design Constraint \*523’.

Note: see work of Stephen Denning etc. 6.Jan.2011

<http://www.stevedenning.com/Business-Narrative/user-stories-applied.aspx>

I think he is copying Cohn too much.

He has a book: “The Leader's Guide to Storytelling: Mastering the Art and Discipline of Business Narrative”, Revised and Updated

See:

<http://en.wikipedia.org/wiki/User_story>

See

http://www.gilb.com/DL461

User Stories paper by Tom and Kai Gilb

In Gilbs' Mythodology Column, Agilerecord.com March 2011

www.agilerecord.com/agilerecord\_06.pdf (whole issue)

**Related Concepts to Story \*665**: Vision \*422 (See Vision Engineering book, and ValuePlanning book), Ambition \*423 (aka the management BS level), Stakeholder \*233

## Strategic Constraints: Concept \*427. August 18, 2001 tg 1923

Strategic Constraints *are constraints we have imposed at our own level, over which we have control.*

*Related concepts:*

*• Fundamental Constraints,*

*• Means Constraints.*

*These are different levels and viewpoints of the same essential ideas.*

*Type: requirement*

*Domain: engineering.specification.requirements.constraints.strategic constraints*

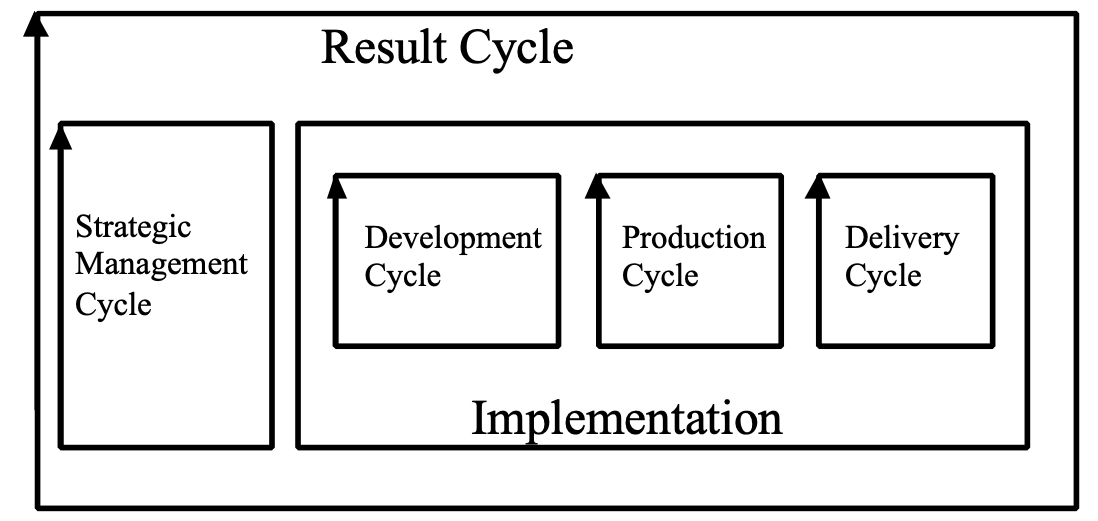
## Strategic Management Cycle Concept \*408 March 31, 2015

A strategic management cycle is a cycle within a result cycle 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.

Notes:

1. It can be thought of in terms of the ‘head’ with the implementation (that is, the development, production and delivery cycles) as the ‘body’ of the Evo result cycle.



Synonyms [Strategic Management Cycle \*408]:

• Evo Management Cycle \*408

• Head (of Evo model) \*408

Related Concept: Result Cycle \*122

Type: Process.

## Strategic objective "strategic objectives" : Concept \*214. August 18, 2001 /April 1, 2003 +MOP MOE. Sept 27 2014 (deleted fundamental in Usage 1)

The objectives which are our main responsibility are called ‘strategic objectives’. They support ‘fundamental objectives’. They are supported by ‘means objectives’.

"The ultimate objectives of the decision-maker" <-[KEENEY92 p.62]

Rationale: this concept is intended to help us focus on our primary responsibility; to not get distracted by things we cannot change (fundamental objectives) and things which are merely intended to support us (means objectives).

Note 1. Strategic objectives are related to a specific level of decision-maker. They are the primary concern and responsibility of that level. They are designed in order to serve that decision-maker’s own boss or higher-authority’s objectives.

*Usage 1. As a result of trying to meet our strategic objectives (‘our ends’), we will plan/design some ‘solutions/means’ which can include both ‘means objectives’, and other ‘strategies’ for meeting our strategic objectives.*

Keyed Icon [Strategic Objectives, \*214]: any requirements or objectives icons can be used to represent the objective directly (like O---->P1---->P2---->>M1--->).

It is @.->.[@] Objective ‘@’, Impacts ‘->’, Fundamental Objective [@]

Related Concepts [Strategic Objective \*214]:

• Fundamental Objective "fundamental objective" . \*229, Fundamental objectives are ‘givens’.

• Means Objective "Means Objectives" : **\*215.** Means objectives are a ‘strategy’, in the format of a performance objective, to meet strategic objectives. They exist only to support meeting strategic objectives.

• Measure of Effectiveness \*486

• Measure of Performance \*482

Synonym: Aim \*001 An ‘aim’ is the ‘total set’ of defined stakeholder-valued requirements, for a defined system, and for a defined level of stakeholder. It is their ends, not their means.

.

"All other objectives of that decision-maker must be means objectives to the strategic objectives" <- [KEENEY92 page 62]

Strategic objectives support ‘fundamental objectives’, and are supported by ‘means objectives’.

“Strategic objectives are intended to define the concerns of a single decision-maker for all decision situations” <- [KEENEY92 page 63]

Type: requirements hierarchy class

Domain: requirements.objectives.strategic

Source: The term ‘strategic objective’ was coined by Keeney [KEENEY92].

## Strategy (noun) "strategy" : (Concept \*047). July 18, 2002, April15 2009

A Strategy is any ‘means’ specified with the intent to impact defined objectives.

If the ‘means’ to satisfy one defined objective is a set of one or more other objectives – which, if themselves satisfied, will contribute to the satisfaction of the *first* objective; then those next level of objectives are also classified as strategies – *from the point of view of* the *first* objective. They are however classified as objectives from the point of view of the strategies defined to satisfy *them. The key to classification is the stakeholder point of view and the process point of view, not the specification itself, generally.*

Source: The conventional dictionary definition of strategy points out that it is a military term relating to positioning forces before a battle so as to have advantage. Current management use of the term is far looser, and includes almost any idea!

“They set only the broadest of objectives and emphasized seizing unforeseen opportunities as they arose…. Strategy was not a lengthy action plan. It was the evolution of a central idea through continually changing circumstances.”

Said by Von Clausewitz in ‘On War’ about Von Moltke and the Prussian General Staff.

Quoted in [Welch01] p448

Related Concepts:

• Strategy is one of many synonyms for ‘design idea’ (\*047, *short form ‘design idea’) which is the main term used in this book to cover all the terms like ‘strategy’ which have in common that they are devices for reaching goals.*

“Intended strategies have no value in and of themselves; they take on value only as committed people infuse them with energy (to paraphrase Selznick, 1957)” <- (“Leadership in Administration”). <- [MINTZBERG94:172]

*Usage: The term strategy is appropriate in business management planning, but not in other kinds of design and planning (engineering for example would have many other preferred terms like ‘design’, ‘architecture’, ‘technique’, ‘component’, ‘structure’). I felt that the word strategy was too specific for general use in this book. But, if your culture uses ‘strategy’ and is comfortable with it, use it as your specific term for what this book calls ‘design idea’ or ‘design’.*

“Nor is there any evidence of a smoothly flowing deductive cascade of strategies. Budgets easily divide and add up arithmetically; strategies do not. How could they, when even the planning literature has failed to distinguish objectives from strategies? Strategies are not sharply-defined entities to be stacked up like crates in a warehouse. They are unique conceptions that only exist in people’s minds.” <- [MINTZBERG94:75]

*To which I feel obliged to add that Impact Estimation tables give us the possibility of adding up the effects of strategies.*

Keyed Icon [Strategy, \*047] : [<Strategy-Tag>]: square brackets symbolizing the solution rectangle drawn icon. It contains a design specification tag, underlined.

Synonyms [Strategy, \*047]: synonyms in essence although culture and emphasis differs.

• architecture, \*192

• means \*047

• Design Idea (\*047), see ‘Idea’ (\*086).

• Design (noun) (\*047) A design is a ‘specified, consciously selected, means’ to reach defined ‘ends’.

• Solution (\*047)

• Strategy. (\*047)

*• Process Improvement Suggestion \*088*

*• Systecture \*546*

Type: design type.

Domain: engineering.design.strategy

## *Stretch >+ Concept \*404. March 15, 2003*

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, scale variables]. 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 to 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.

Synonyms [Stretch \*404]:

• Stretch Target \*404

• Stretch Level \*404 (see Level \*337)

• Outstanding [Intel, 2003]

Related Concepts [Stretch \*404]:

• Need \*599

• Target \*048

• Goal \*109

• Budget \*480

• Wish \*244: Stretch is different from a Wish level specification in that a Wish can be arbitrarily high (whatever seems useful value to a stakeholder) without regard to costs or even technical possibility (They can’t know).

Keyed Icon [Stretch \*404]: >+

In context: ---->+--->O---->+---->

Type: Parameter, Target.

Historical Note: This concept was first used in Planguage by Pete Fuenfhausen, then at Nokia, Dallas, TX, September 1999.

“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’

[WELCH01, pages 385-6]

## Structure: Concept \*512. November 21, 2001

A structure is the observable implementation of a system component. The way it is built.

The structure of a component, process or system explains its performance and cost attributes. The repetition of a structure in a new system is generally expected to result in the same or similar performance and cost attribute impacts as when impacts were observed in other systems.

Related Concepts [Structure \*512]:

• Component \*022

• Design Structure {\*047 + \*512}; an observable attribute of a system – the detailed structure of a design of anything including hardware, software, specifications, ideas. Organizations, biological systems, nature, and contracts.

• infrastructure, the basic structure of a defined system. Infra- : below, beneath

• design \*047: this is the idea of the structure.

• impact \*087

Type: system.descriptor

Domain: system.

## Study [PDSA] Concept \*171 January 21, 2003 B

Study [PDSA] is part of the Plan Do Study Act Cycle (See Plan Do Study Act Cycle \*168). 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.

The ‘Study’ phase is a preparation for drawing conclusions and taking action in the ‘Act’ phase.

*“Study the results, what did we learn, what went wrong?”*

W. E. Deming [Personal Communication 1991]

In Evo, the ‘Study’ phase (P3 in **Procedure.SM)** is the equivalent of the more general PDSA concept ‘Study’. Study in Evo 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).

Synonyms [Study [PDSA] \*171]:

• Check (and then ‘PDCA’ Cycle). Note this variation is widely spread, and was even used by Deming in his book ‘Out of The Crisis’. However, he preferred ‘Study’, as Shewhart had originally taught him, to the term ‘Check’. ‘Check’ had other meanings he did not like, such as to ‘hold back’ (See also Plan Do Study Act Cycle \*168).

Related Concepts [Study [PDSA] \*171]:

• Plan Do Study Act Cycle \*168

• Plan [PDSA] \*169

• Do [PDSA] \*170

• Act [PDSA] \*172

• Evolutionary Project Management \*355

• Statistical Process Control (SPC) \*466

Keyed Icon [Study [PDSA] \*171]: There is no keyed icon for Study [PDSA].

Suggestion ‘?]’. A rough draft for the icons [January 21, 2003, TG]:

Plan [> left side of process rectangle and a Goal ‘>’ symbol.

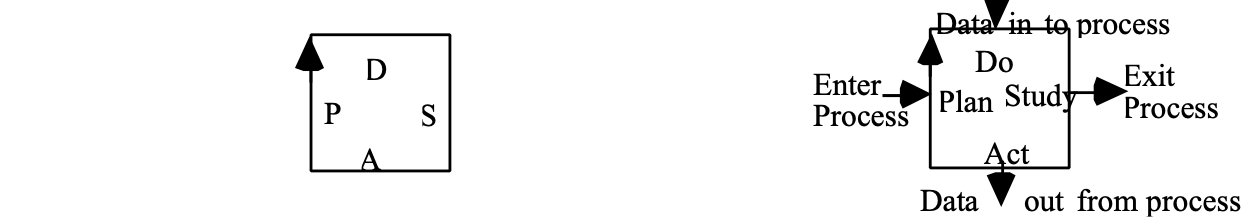
Do ^! Top side of process rectangle and an action symbol ‘!”

Study ?] right side of process rectangle and an an inquisitive symbol ‘?’.

Act V± bottom side of process rectangle and a take it or leave it symbol ‘±’.

Abbreviation [Study [PDSA] \*171]: : ‘S’

Drawn Icon [Study [PDSA] \*171]: Study is the east side of the rectangular process symbol.



The four corners of the process icon stand for the Shewhart process-cycle definition of ‘Plan Do Study Act’.

Type [Study [PDSA] \*171]: Process.

Domain [Study [PDSA] \*171]: SPC.

## Sub-Process Concept \*173 February 21, 2003

A sub-process is a process contained in a supra-process.

Type [Sub-Process \*173]: Process.

## Subjective [adjective] Concept \*657

Dependent on defined stakeholder value judgment. Not ‘Objective’

For example:

Subjective Quality:

The subjective formal and informal position adopted by or for defined stakeholders, regarding the utility or value of a particular objective quality level.

“Stakeholder X has specified that less than 99.98% availability for the software is useless in practice.”

Related Concepts [\*657]:

• Objective [Adjective] \*656

• Subjectivity \*387

## Subjectivity: Concept \*387. August 20, 2001 tg 1520

Subjectivity is that which is *predominantly* a matter of personal opinion, or taste, or state of mind.

Note 1.

In all forms of analysis and estimation there is some component of personal filtering and opinion, or cultural bias or prejudice. This can even be present in apparently subjective and automated measuring processes, in the form of selection of the subject to be measured, and evaluation of the data gathered. We therefore do not recognize absolute objectivity as a reality; outside of some ideal. Not do we recognize absolute subjectivity as a reality, outside perhaps of the theoretical notion of a random number as an answer.

Subjectivity is the degree that the process is biased by any number of factors related to people and culture. Objectivity is the degree that the process of analysis or estimation is repeatable, and that it will come up with essentially the same answers irrespective of individuals or cultures doing the process.

‘Subjectivity’ is a term usually used in a negative way. For some very important analytical processes (e.g. electors, consumer choice, market analysis, Gallup Polls) it is precisely the subjective opinion which is the critical thing to measure (objectively, of course!). There is no objective Physics-like science about people’s opinions. But there is a science in understanding people’s opinions, by which all political and economic power in the world is derived.

The degree to which the ‘degree of subjectivity’ is interesting for you depends on the objectives you have with your analysis.

If it is to determine the learning time needed for a task, then objectivity is most central. But of course subjectively we must decide that this measure is interesting, decide on the nature of the task and experimental measurement process; and finally decide on the nature of people and selection methods for people to be measured doing that task. In short, there is a lot of subjectivity in objective measurement processes.

The essential idea behind any discussion of an analytical process’ objectivity or subjectivity is the worry that too much objectivity may somehow obscure the ‘absolute truth’ (whatever that is).

Rather than simply classifying something as ‘objective’ or ‘subjective’ we should back up our opinion, and communicate it to other people by somehow listing the factors and worries we have, rather than hiding them and leaving the reader to guess at our knowledge or opinion.

Planguage strongly encourages the specification of all factors that will help the reader understand the subjective and objective influences which led to a specification.

Antonym: Objectivity.

Type: specification classification. (subjective opinion).

Rationale [Subjectivity]. This is a central point for Planguage and this book.

## Subset Concept \*222, February 4, 2003 tg

A specified, or defined, part of a specified, or defined, larger whole.

Example:

Goal [USA]: 60%.

‘[USA]’ is a subset of the Place scope of the system. This Goal only applies to this subset.

Related Concepts [Subset \*222]:

• Consists Of \*616

• Component \*022

• Includes \*391

Type [Subset \*222]: Noun [Relationship].

## Success Range. Concept \*548 November 24, 2004

A success range is meeting the goal level or better.

The success range is a target area, or landing zone, that extends from a specified goal level, to any better level, and includes any stretch or wish levels.

Note: A success range is one or more ranges, depending on qualifiers, on a defined scalar attribute which are at or above the Planned level (Goal) and below any upper Catastrophe Limit – that is also to say within any Survival Limit. It includes relevant Stretch levels and any relevant Wish levels which are above the Goal level. It specifically does not include the Acceptable Range below the goal level.

Using keyed icons to relate success range to other defined concepts:

••••••••••[!!!!!!!!!------->===>+====>?===]•••••

••••••• is Catastrophe Range, [ and ] survival limits, !!!!!!! is Failure Range, ======= is Success Range.

> = Goal, >+ is Stretch Target, >? Is Wish Target (better than the Goal level).

Related Concepts:

• Failure Range \*546 Never in a success range.

• Catastrophe Range \*603 Never in a success range

• Acceptable Range \*545 Not a Failure or Catastrophe Range, but not as good as the Goal level.

• Survival Limit \*440 Defines the limit of a success range.

• Landing Zone \*605

Keyed Icon [Success Range, \*548] >==========>

Where the lower limit of the success range is meeting the specified Goal (left >) level. The upper limit of the success range is denoted by the best extent of the attribute (infinity, 100% or 0% as the nature of the attribute may dictate) as iconically indicated by the scale attribute arrow tip at the right hand side of the icon above (>===**>**).The success range specifically includes stretch and wish target levels.

Success Range concept suggested by L. Brodie Spring 2002 in her doughnut diagram.

## Sum of Benefits Concept \*008 January 20, 2003 B

Within an Impact Estimation table, a ‘sum of benefits’ is the total quantitative impact on the specified performance targets made by a specific design idea (or set of design ideas or Evo step).

For a specific design idea, it is the sum of its percentage impacts (%Impacts) on all the performance attributes.

Example:

|  |  |  |  |
| --- | --- | --- | --- |
| Design IdeasRequirements | Architecture X | Package Y | Service Z |
| Reliability | 30% | 25% | 10% |
| Usability | 60% | 10% | 0% |
| Portability | 50% | 33% | 16% |
| **Sum of Benefits** | **= 140%** | **= 68%** | **= 26%** |
| Cost | 6% | 20% | 30% |

*For each individual design idea (Architecture X, Package Y and Service Z), the percentage impacts for all the performance requirements are summed. This gives an idea of the total contribution that the design ideas each make relative to one another, towards satisfying the stakeholder requirements.*

Notes:

1. A Sum of Benefits is used together with the associated Sum of Costs \*128 to compute a ‘benefit to cost ratio’ for a design idea. This is useful for evaluating ‘designs’ and ‘steps’ against their alternatives.

Synonyms [Sum of Benefits \*008]:

• Sum of Performance \*008

• Benefit Impact \*008

Keyed Icon [Sum of Benefits \*008]: ∑.O+

Type [Sum of Benefits \*008]: Metric.

**Domain [Sum of Benefits \*008]: Impact Estimation**.

## Sum of Costs Concept \*128 June 6, 2003

Within an Impact Estimation table, for a specific design idea (or set of design ideas or Evo step), the sum of costs is the vertical sum of the percentage impacts of selected scalar *resource* attributes. Sometimes, the numeric *average* is calculated by dividing the sum by the number of summed resource attributes.

Notes:

1. This gives a general idea of the resource consumption by a specific design idea (or set of design ideas or Evo step), and can be compared to the performance improvements likely to be obtained.

2. Performance to cost ratios can be calculated by dividing the Sum of Costs into the Sum of Performance.

3. Value to cost ratios can be calculated by dividing the Sum of Costs into the Value calculated/estimated for stakeholder value.

4. An alternative to using the Sum of Costs in a ‘ratio’ calculation is to use the sum of the absolute financial costs.

Synonyms [Sum of Costs \*128]:

• Resource Impact \*128

Related Concepts [Sum of Costs \*128]:

• Sum of Performance \*008

• Performance to Cost Ratio \*010

• Value to Cost Ratio \*635

Keyed Icon [Sum of Costs \*128]: ∑.-O

Type [Sum of Costs \*128]: Metric.

## Sum of Performance Concept \*008 June 6, 2003

Within an Impact Estimation table, the ‘sum of performance’ is the total quantitative impact on the specified performance targets made by a specific design idea (or set of design ideas or Evo step).

For a specific design idea, it is the sum of its percentage impacts (%Impacts) on all the performance attributes.

Example:

|  |  |  |  |
| --- | --- | --- | --- |
| Design IdeasRequirements | Architecture X | Package Y | Service Z |
| Reliability | 30% | 25% | 10% |
| Usability | 60% | 10% | 0% |
| Portability | 50% | 33% | 16% |
| **Sum of Performance** | **= 140%** | **= 68%** | **= 26%** |
| Sum of Costs | 6% | 20% | 30% |

*For each individual design idea (Architecture X, Package Y and Service Z), the percentage impacts for all the performance requirements are summed. This gives an idea of the total contribution that the design ideas each make relative to one another, towards satisfying the stakeholder requirements.*

Notes:

1. A Sum of Performance is used together with the associated Sum of Costs to compute a ‘Performance to Cost Ratio’ for a design idea. This is useful for evaluating design ideas and Evo steps against their alternatives.

Related Concepts [Sum of Performance \*008]:

• Sum of Costs \*128

Keyed Icon [Sum of Performance \*008]: ∑.O+

Type [Sum of Performance \*008]: Metric.

## Sum for Requirement Concept \*340 January 20, 2003

For a given set of design ideas, the *sum* of their estimated-or-measured percentage impacts on a single performance or resource attribute.

The right-hand column of an Impact Estimation table usually contains this sum for an attribute. It is a rough measure of how likely the requirement is to be met.

Synonym [Sum for Requirement \*340]:

• Sum Requirement \*340

Keyed Icon [Sum for Requirement \*340]: ∑.[@]

(Sum, and Requirement (Target @ and Constraint [ ] ).

Related Concepts [Sum for Requirement \*340]:

• Percentage Impact \*306

• Safety Factor \*131

• Safety Deviation \*405

Type [Sum for Requirement \*340]: Metric.

Domain [Sum for Requirement \*340]: Impact Estimation.

## Supplementary Designs \*589 July 16, 2002

*Supplementary* designs are needed to move the set of designs towards the design Goal levels, other target levels or towards meeting other requirements (constraints, functions).

Related Concepts [Supplementary designs, \*589)

• alternative designs, \*588 cannot supplement, they are mutually exclusive

## Support System Concept \*152 April 1, 2003

A support system is any system that has performance levels that impact a defined stakeholder environment. A support system is intended to contribute to the stakeholder benefit level.

Related Concepts [Support System \*152]:

• stakeholder \*233

• measure of effectiveness \*486

• measure of performance \*482

• benefit \*009

• performance \*434

• supports \*415

Type [Support System \*152]: Environment

## Supported By Concept \*414 February 18, 2003

‘Supported By’ is used to list the tags of any and all specifications that contribute usefully to the accomplishment of a defined requirement.

Note:

1. The specifications that can provide support include resources, functions, designs, design ideas and Evo steps.

Example:

Goal X:

Scale: <some scale definition>

Goal [Next Version]: 55%.

Supported By: Design Idea {A, B, C}.

Related Concepts [Supported By \*414]:

• Supports \*415

• Impacts \*334

• Impacted By \*412

Type [Supported By \*414]: Parameter [Relationship].

## Supports Concept \*415 February 18, 2003

‘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 [Supports \*415]:

• Impacts \*334

• Supported By \*414

Keyed Icon [Supports \*415]: ->>

“Icon is similar to the Impacts symbol, ‘->’, but with additional emphasis, ‘>>’.”

Examples :

Design X ->> {Availability, Capital Cost}.

Function Z: ->> {Function XYZ}.

Example:

Screen Design: {Icons, Windows} ->> {Usability. Intelligibility, Portability}.

Type [Supports \*415]: Parameter [Relationship].

## Supra Concept \*264 February 23, 2003

‘Supra-‘ is a prefix, which can be attached to *any* component type that has its own sub-components (kids).

Notes:

1. Terminological applications of ‘Supra’ include: {supra-process, supra-function, supra-objective, supra-design, supra-step}.

2. ‘Supra’ may be used without the hyphen: ‘supraprocess’. Normal usage is to use a hyphen. Use without a hyphen might give your spelling checker some problems!

Related Concepts [Supra \*264]:

• Parent \*267: ‘Supra’ differs from ‘Parent’ in that it can be ‘supra’ to ideas *at any level of detail below it*, while a parent can only be related to its *immediate* kids.

• Kid \*266

• Subset \*222

Type [Supra \*264]: Prefix [Relationship].

## Supra-process "supra-process" : Concept \*257. August 20, 2001 tg 1550

A process that contains other defined processes.

Related Concepts:

• See mega-process which means a large process, but says nothing about a hierarchy of processes.

• Supra- \*264

Type :process specification relationship concept

Domain: engineering.specification.relationship.hierarchy.supra-process.

## Survival Concept \*440 October 30, 2004

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, observation of human tolerances, 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. Any level worse an edge of a Survival Range is a ‘catastrophe’ level.

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).’ While, Survival sets the upper financial limit to avoid disaster.

Synonyms [Survival \*440]:

• Survival Level (see Level \*337)

• Survival Limit (see Limit \*606)

Related Concepts [Survival \*440]:

• Fail \*098

• Limit \*606

• Catastrophe \*602

Keyed Icon [Survival \*440]: [ ]

“The ‘[‘ being a lower limit and the ‘]’ being an upper limit.”

Type [Survival \*440]: Parameter [Scalar], Constraint.

## Survival Range: Concept \*585 March 3, 2003

A Survival Range is the numeric range of a scalar attribute that indicates survival of the system, with respect to that attribute alone.

Keyed icon [Survival Range, \*585] ---[======]------->O------[=====]---->

*Note: the ‘[‘ being a lower limit and the ‘]’ being an upper limit*.

*The ‘[=====]’ is the Survival Range*.

Related Concepts:

• Catastrophe Range [ \*603 •••• ] the range just outside the Survival Range

Type [Survival Range, \*585]

## Symbol Concept \*162 February 27, 2003

A symbol is *any* *defined* representation of information, concrete realities, or abstractions such as emotion.

A symbol is defined as a representation of some defined information or other type of object.

A symbol is different from an icon in that all icons are graphic symbols, but a symbol in general can include any *non-pictorial* representation of information, such as text and words - or even sounds and colors.

Examples of ‘Symbol’ in Planguage include things such as:

• a parameter (‘Goal’), all Planguage parameters are defined symbols for concepts.

• a defined term (‘Sales to Date’),

• a ‘convention’ like Capitalized Tags are a symbols that Capitalized terms are defined in this project, or in this system of documentation. All Planguage grammar conventions are symbols.

• Symbol also includes the notion of Planguage pictorial (graphic) icons (keyed and drawn).

Related Concepts [Symbol \*162]:

• Icon \*161: All icons are symbols, but not all symbols are icons.

• Planguage Concept \*188

Type [Symbol \*162]: Specification Concept.

## Symbology "symbology" : Concept \*209. August 20, 2001 tg 1555

The system of symbols representing ideas, which includes {terminology, icons, symbols}.

Note 1. Planguage is rich in symbols for representing ideas about systems.

Related Concepts:

• Icons \*161 A graphic symbol, keyed or drawn, representing a term in Planguage. Synonym ‘symbol "symbol" ‘.

• Keyed Icons \*144, A {non-English, nor any other spoken natural language} set of key-able characters, with agreed defined interpretation.

• Concepts \*188. A Planguage concept is a formally specified idea which can use any number of devices to reference the concept definition.

• (Concept) Glossary: example the one you are looking at now for Planguage.

• Planguage \*030. Planguage is a language and a set of related methods for systems engineering.

Type: general concept of prime interest to Planguage.

## Systecture © Concept \*564 May 27, 2003

See Systems Architecture \*564. Systecture is a conjunction of the term ‘system architecture’.

Historical Background [Systecture \*564]: 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](http://www.systect.com) (Systect, Inc. ‘The system architects’) a systems architecture company, but no use of Systecture at all. © [Tom@Gilb.com](mailto: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*[[16]](#footnote-16)*.

## Systect: Concept \*565. July 19, 2002

A systect is a person who does Systecture (systems architecture) – a systems architect. It is a conjunction (systems architect).

Related Concepts [Systect, \*565]

• Systecture \*564 systects do systecture

• architect \*193 systects are systems architects

• designer \*190 all systects are designers, but not all designers are systects

Background: See background for Systecture above.

A web search turned up [www.systect.com](http://www.systect.com) (Systect, Inc. ‘The system architects’) a systems architecture company.

## System [Planguage] Concept \*145 May 4, 2003 B

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. Svein Øvergaard, 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”

ISO 9000

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, /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 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.

An integrated composite of people, products, and processes that

provide a capability to satisfy a stated need or objective.

system

[MIL-STD 499B]

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 [Planguage] \*145]:

• System \*145

• Object \*099: Separate concept number allocated as the terms tend to be used distinctly.

Related Concepts [System [Planguage] \*145]:

•Attribute \*003

Keyed Icon [System [Planguage] \*145]:

[ ----->O----> ]

or

----->(<*mission tag*>)----> “A system with resources, function and performance. Conditions are implied.”

Example:

---->(Mobile Phone)---->

In this context the mission tag, ‘Mobile Phone’ can serve as the reference name for the system, the attributes of the system being implied by the arrows.

Type [System [Planguage] \*145]: Systems Engineering Concept.

## System Descriptor: (Concept \*384). October 28, 2001 1513

A system descriptor is any external or internal attribute {performance, quality, cost, function, constraint, design or other notion} which is used in some way describe or characterize a system.

Note 1: this term is intended to be all inclusive of any descriptive attribute, characteristic, property view or stakeholder value of a system which it would be useful to include, even if our current imagination does not seem to include it

Synonyms: system characteristic, system property, system attribute, system spec

and (partially) system quality, system performance system cost, system function

Related Concepts:

• attribute \*003. the multiple characteristics of a system.

• system \*145

• specification \*137. A ‘specification’ communicates one or more system ideas and/or descriptions to an intended audience.

Type: System specification components, or system descriptors

Domain: engineering.system.specification.

The distinction between attribute (\*003) and a descriptor

• descriptor: something actively used to describe a system

• attribute: something which is in the system ( whether used to describe it by someone or not)

## System Description Language (SDL) "System Description Language" : Concept \*183. March 4, 2003

SDL is the subset of Planguage for describing systems, including ‘processes viewed as systems with attributes’. This specifically excludes subsets of Planguage used to describe project plans which are not the system itself (for example Evo and SQC).

Abbreviation [System Description Language \*183]: SDL.

Related Concepts [System Description Language \*183]:

• Basic Planguage/Planguage Generics \*225, Basic Planguage.

• Process Language \*246 Planguage which describes processes.

• User-Defined Term \*530

• Project Language \*247: the subset of Planguage selected for a given project.

• SQC: a subset of Competitive Engineering whose Planguage concepts are not a part of the System Description Language. They deal with specification quality control.

•  Evo: a subset of Competitive Engineering whose Planguage concepts are not a part of the System Description Language. They deal with Evolutionary project management.

Notes:

System Description Languag includes all possible language elements, including those not chosen or applied in a particular project (PL)

System Description Language includes everything available in Planguage from all sources. The Project Language is the subset actually used by a defined project.

Type [System Description Language \*183]: Specification.

## Systemic Concept \*262 November 21, 2002

Systemic means ‘of’, or ‘relating to’, the entire system.

This word is especially useful in connection with process improvement by removing root common causes. The root causes (systemic causes) are the deeper causes of defects in the system that it pays off to attack.

There may be a series of related systemic causes of which only the earliest one would be the ‘root’ cause.

Note: Systemic is not the same as ‘systematic,’ which means regular purposeful process.

Related Concepts:

• Root Causes \*263

• Process Improvement \*114

• Chance Causes \*019 (=special causes)

• DPP \*042

• Statistical Process Control \*466

Type:

Domain: Statistical Process Control, Continuous Process Improvement.

## System Definition Rule Concept \*518 January 14, 2003

A system definition rule is any statement which specifies a rule of any kind for the design or action of any system.

Type: Parameter

Example:

Function-A

Description: <describe function A>

R1: Rule: The Customer Number has 12 characters excluding punctuation.

R2: Rule: Rule About Signing " this is a reference to a rules defined elsewhere"

These 2 rules are attached to a function definition.

Related Concepts

• specification rule \*609 a specification rule is a subclass of system definition rule which is limited to rules of how system specifications shall be done.

## Systems Architect Concept \*193 July 23, 2003

A systems architect is a person or group, who carries out the work tasks of systems architecture (a process).

Notes:

1. The systems architect, the agent for the systems architecture process, is the technical interface between the customer and all the technological means for helping the *stakeholders* reach their objectives.

2. The systems architect manages all other necessary technologies, and is empowered to choose the best alternative technology areas to solve ‘the problem.’

3. The systems architect must also ensure suitable interfaces between the technologies that must be integrated.

4. The specifications of the systems architect will have higher priority than design ideas originating at lower levels of systems engineering specification.

5. The role of systems architect might be carried out by an individual, or it could be a series of people through time, or it might be a collective responsibility. It can be a part-time responsibility.

6. Architect is from ‘Archi-Tecton,’ which means ‘Master Builder’. ‘Archi’ is not from ‘Arch’, but from ‘Arche’: primitive, original, primary[[17]](#footnote-17).

I think architecture is the science of structure and the structure of whatever is, whether it is music, whether it is painting or building or city planning or statesmanship’

Frank Lloyd Wright, An Autobiography, Horizon Press, NY, 1977 (See page 131)***[[18]](#footnote-18)***

Related Concepts [Systems Architect \*193]:

• Systems Architecture \*564

• Designer \*190

Type [Systems Architect \*193]: Role.

## Systems Architecture Concept \*564 July 23, 2003

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 [Systems Architecture \*564]:

• Systecture \*564

• Architecture \*192

Related Concepts [Systems Architecture \*564]:

• Architecture Engineering \*499

Type [Systems Architecture \*564]: Architecture.

<http://www.sei.cmu.edu/architecture/community_definitions.html>

is a link with a great many opinions about the meaning of software architecture. Nov 22 2007 tg

## Systems Engineer: Concept •574 July 12, 2002

One who practices systems engineering (\*233) or is qualified to do so.

## Systems Engineering Concept \*223 July 17, 2003

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.” [BLANCHARD98]

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 [FAA98].

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.

*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.

Figure \*223: [Editor Note to Publisher: This diagram had to be imported via PDF???]

Related Concepts [Systems Engineering \*223]:

• Engineering \*224

• Architecture Engineering \*499

• Requirement Engineering \*614

• Design Engineering \*501

• Evolutionary Project Management \*355

Type [Systems Engineering \*223]: Discipline [Engineering].

## Tag Concept \*146 January 16, 2003

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.

Examples:

Reliability [USA, Retail Dealers].

&

See Project Oasis.Requirements.Usability.Fail [USA, Retail Dealers].

Tag Typography Rules:

1. Tags must always begin each alphabetic word with a Capital Letter.

2. User-defined tags are not allowed to be already defined parameter terms in Planguage (such as Fail or Rationale).

3. Any useful typography including Bold, Underline, ALL CAPITALS, color, or italics may be used to emphasize tags.

Synonyms [Tag \*146]:

• Identifier

• Name (informal use). The name of a specification.

Type: Optional Parameter.

Type: Optional Parameter.

## Tailored Planguage "Planguage Specifics" : Concept \*226, August 22, 2001tg 2310

Any modifications to a defined version of Basic Planguage (\*225) to tailor it to a specific environment.

*Note: These could be additions, reduced subsets or modifications to Basic Planguage. These could be specified either as a total tailored variation of Basic Planguage, or as specific modifications to an existing reference standard of Basic Planguage.*

*Examples: translations, variants, extensions, and subsets.*

## Target Concept \*048 January 23 2008

Targets are a class of Requirements that we seriously plan to *reach*. They contrast to Constraints that we seek to *avoid; and Benchmark Levels which are merely analytical information, without formal commitment to reach them.*

Notes:

1. NOT CONSTRAINT. 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. Function targets, or ‘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 – but can have specified ones (‘Rationale’ and other attached specs): nevertheless, they are considered required by some stakeholder.

3. CONTENT. A scalar target specification consists of a numeric value (its target level) and its relevant [qualifiers] and a priority-level classification of Goal, Budget, or Stretch.

Wish, Ideal, and Perfect scalar levels are initial reference points, and are classified as Benchmarks. They are not targets, unless the levels they refer to are also formally acknowledged as such, by being specified as Goal, Stretch, or Budget.

4.COLLECTIVE USE. 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.

5. AIM NOT AVOID. A target is distinguished from constraints in that we are aiming to reach, at least to, the target levels, or to deliver the function target; while scalar constraints are requirements levels we want to avoid. Function targets are functions we want to deliver, while ‘function constraints’ are functions we want to avoid.

6. PRIORITY: Target level requirement specifications can be expressed by the parameters Goal/Budget and Stretch. Each one of these indicates a different level of commitment and priority. Benchmark levels (Past, Record, Trend, Ideal, Perfect) can be thought of as indicating a weaker level of stakeholder interest or value, if any. Constraint levels (Fail, Catastrophe) are a stronger level of priority than Target levels.

7. SPECIFICATION ACTIVATION. Target specifications can become committed requirements (become ‘prioritized’, or ‘activated’),

• when their qualified conditions are true, (especially in the case of Goal) and

• when it is economically and technically possible (in the case of Stretch)

• when other higher priority targets are fulfilled, and there is still resource available to meet them (this applies to Stretch and Wish, as well as lower priority Goal requirements).

8. There are two kinds of target: ‘scalar’ and ‘binary’.

9. Scalar targets can be expressed using the parameters {Goal, Budget, Stretch}.

10. Binary targets are function targets.

Related Concepts [Target \*048]: • Goal \*439: a performance target• Budget \*421: a resource target

• Benchmark \*007 updated Jan 23 08

• Constraint \*218

• Function Target \*420

• Function Constraint \*469

• Scalar Target \*470

• Stretch \*404

• Wish \*244

• Requirement \*026

Keyed Icon [Target \*048]: @

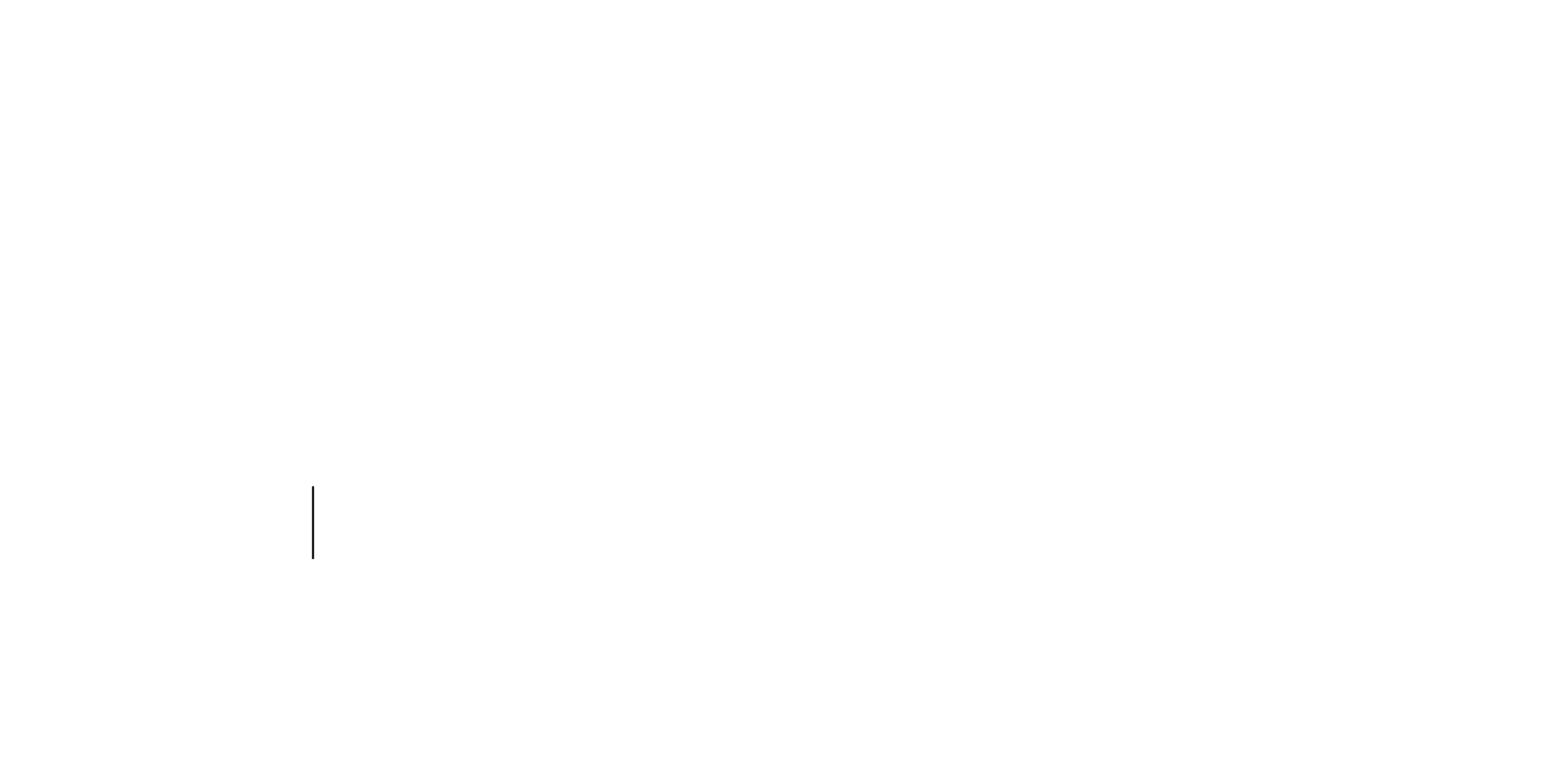


Figure [\*048] Scalar targets can be specified for both performance and resource requirements.(new diagram Jan 23 08)

Type [Target \*048]: Requirement Level Specification Type.

## Target System: Concept \*610. January 19, 2003

A target system is a defined system that is the target of a defined change effort by an agent system.

A target system is the one that evolutionary steps are delivered to. A target system is the one that has specified performance and cost targets that we are trying to reach.

Related Concepts [Target System, \*610]:

• System, \*145 the general concept of a system

• Agent System \*611 a system used to change a defined target system

Type: Noun, Specification Type

## Task Concept \*149 May 20, 2003

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 [Task \*149]:

• Act [PDSA] \*172

• Process \*113

• Procedure \*115

Figure \*149 : A process with its three main components.

Type [Task \*149]: Process Component.

## Team Leader [SQC] Concept \*281 November 20, 2002

A team leader is responsible for managing a specification quality control (SQC) process.

Notes:

1. The team leader must know the SQC process thoroughly, and help the team members to perform. The team must follow the ‘best-practice’ SQC processes. An SQC team leader is normally trained (for about a week) and formally approved, as suited to practice.

2. A team leader is not a management, or project management, or general-leader role.

Synonyms: {SQC Team Leader, SQC Leader, Moderator (IBM Inspection term)}.

Type: Role.

Domain: SQC.

## Technology: Concept \*352. August 22, 2001 tg 2317

The methods and materials used to construct or modify a system. The design ideas and functionality.

## Template Concept \*254 February 22, 2003tg

A template is an example or ‘model’ of something, which can be used to help people to tailor or make something, based on that model.

Notes:

1. Templates are one method for transmitting knowledge and culture through time and space. They are collections of knowledge.

2. An alternative way to describe a template is as a ‘form’. A template is usually for a specific document type or purpose. However, a template is intended as a list of ideas, or a pattern for similar specification. A form is something intended for direct data collection. It is fixed in format and appearance.

3. There is a use of the term ‘template’ as a kind of a computer form, which is filled out on a screen.

Example:

Template [Function]:

Version: 17-Mar-2001.

<Function Tag 1>:

Type: Function.

Description: <describe the function here, well enough to allow testing of it>.

Attribute R: Scale: <?>. Goal: <?>.

{Attribute S: Scale: <?>. Goal: <?>.}

Fuzzy brackets (<…>) are used in a template to indicate what to ‘fill out.’

{Set parenthesis} are used to indicate optional specification types.

Example (of filling out the template):

Flagship Product:

Type: Function.

Description: Provide a mobile telephone service. Product code 9998.

“This describes the function which has an explicit and local (this definition) for the 2 qualities below. Other attributes and specifications may be implied by other specifications elsewhere.”

Reliability: Scale: Mean Hours between Faults. Goal [End Dec 2001]: 22,000.

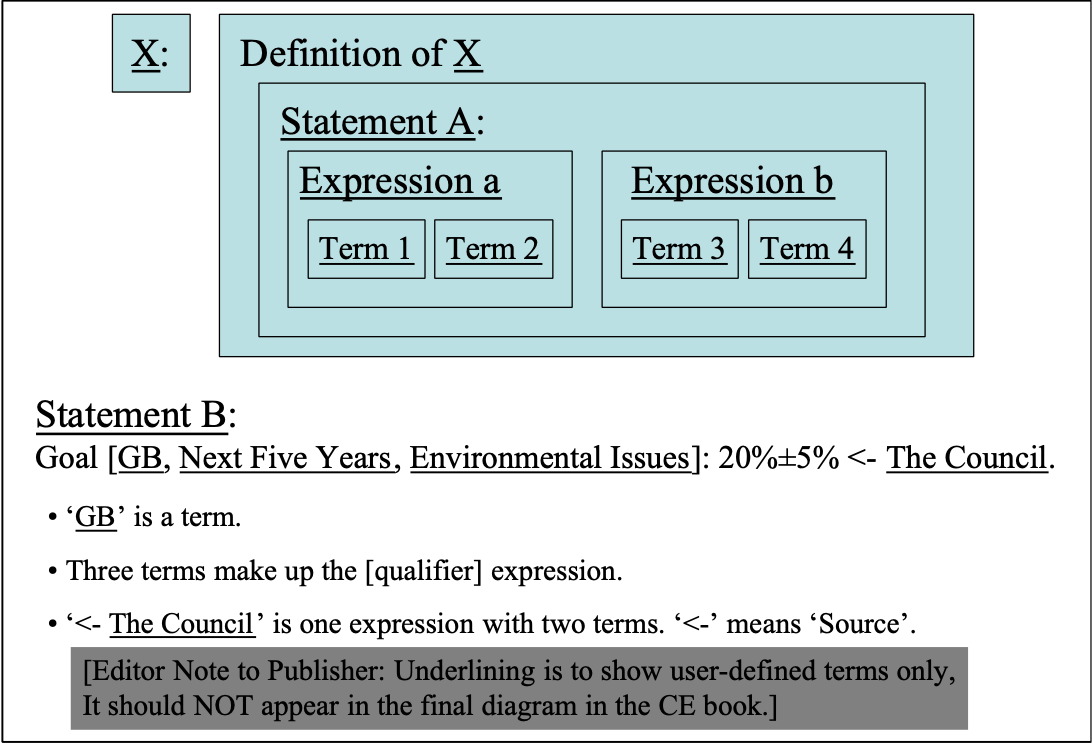
Battery Life: Hours Life. Goal [Standby]: 500. Goal [Calling]: 50.

Related Concepts [Template \*254]:

• Form \*068

Type [Template \*254]: Guidance.

## Term Concept \*151 May 18, 2003

A Planguage term is a minimum set of words and/or symbols (icons) that has a *defined* Planguage meaning.

Notes:

1. ‘Term’ points us to a specific defined concept. A ‘term’ may be tagged, with one or more tags, or it may be untagged. A ‘term’ may itself be a tag or a defined parameter.

The following are terms:

Goal (a parameter, defined in Planguage)

Tag 23 (a tag, a user-defined term, defined somewhere in project documentation)

60% (a numeric value, defined by Scale associated with a parameter)

🡨 (a Planguage-defined keyed icon, for ‘Source’)

The following are not defined ‘terms’:

Summary of Contract (a paragraph heading)

the next person (a non-defined (not ‘capitalized’) phrase in the documentation)

<improved> (a fuzzy word, marked by fuzzy brackets)

2. ‘Terms’ as tags are used to refer to {expressions, statements and definitions}. They name concepts, and give us a way to reuse a term in many places.

Related Concepts [Term \*151]:

• Statement \*140

• Expression \*302

• Definition \*045

Type [Term \*151]: Specification Concept.

## Terminology [Planguage]: Concept \*210. August 22, 2001tg 2324

The vocabulary of technical terms used and defined in Planguage, and in projects which make use of it.

## Test Concept \*256 March 31, 2003

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, or it 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.

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 [Test \*256]:

• 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).

Type [Test \*256]: Process, Parameter.

## Threat: Concept \*671. 140814, October 22, 2001tg

A ‘threat’ is something that can *potentially* cause some degree of project failure, lack of success or negative consequences.

It is distinguished from an Attack, which is a *successful* penetration of the Threat into a system. The Threat has ‘materialized’ in practice.

It is distinguished from a Risk, which is a result of the combined effect of a Threat/Attack and the corresponding defenses (Mitigation)

*Threat analysis is any process that identifies and prioritizes threats. Gap analysis is one type of threat analysis: restricted to analysis of residual gaps between current accomplishment and defined targets.*

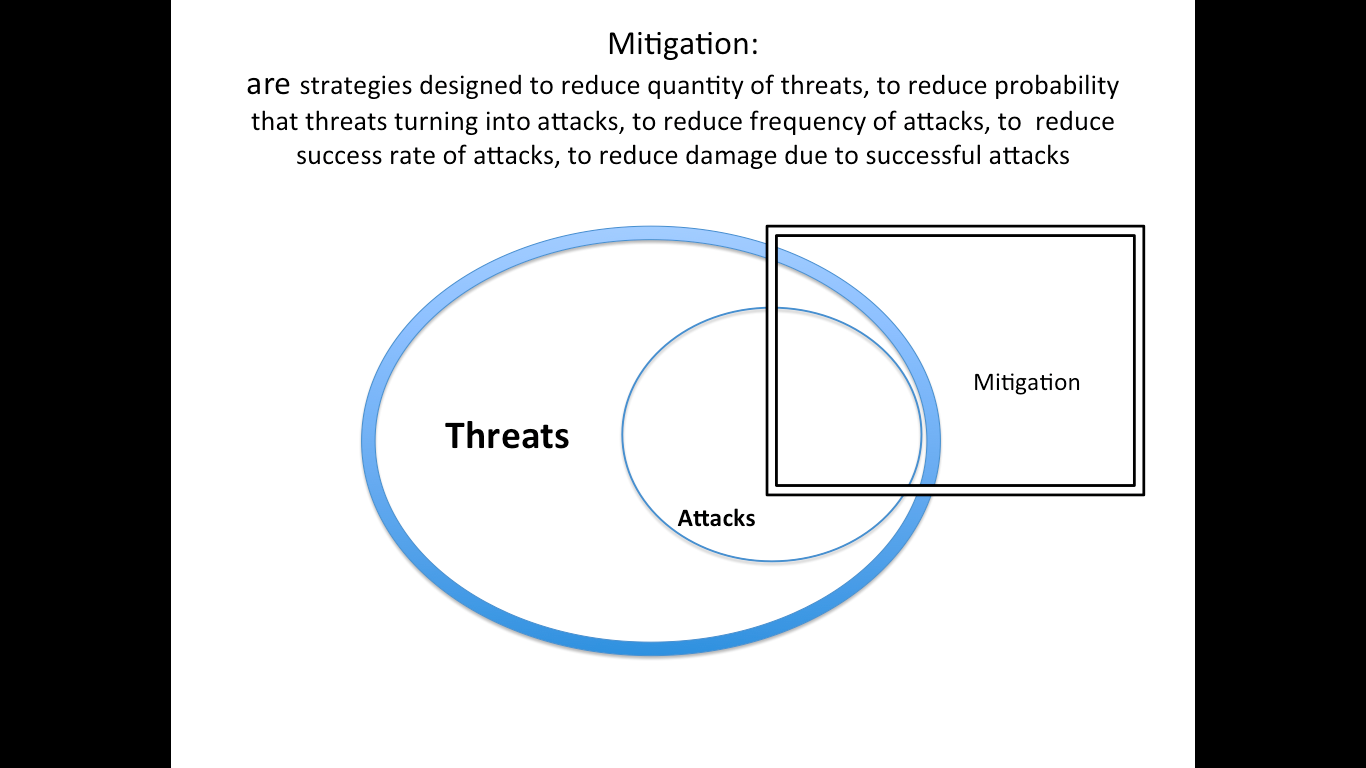


Diagram \*671: Threats are potential attacks on a system. Some threats turn into real attacks, threatening Damage to a system. Mitigation (built-in and post damage) are strategies designed to thwart attacks, or reduce them and their damage.

**Synonym**: Potential Attack

**Related Concepts:**

Attack \*672

Mitigation \*670

Risk \*309

**Type**: Parameter

## Time Concept \*153 May 4, 2003 B

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.

Examples:

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%.

Type [Time \*153]: Parameter, Scope Concept.

## Time Between Successive Products: Concept \*361. August 22, 2001tg 2330

The TBSP is calculated by subtracting the release date the earlier product from that of the later product, giving calendar days difference

Abbreviation: TBSP.

Synonym: Time To Market Prime (TTM’)

Source: JANDOUREK96.

*Note : In the case of serial independent products the TBSP is equal to the TTM (Time To Market) of the last product. In other words this measure gets interesting when you try to hit the market more frequently than the length of a product development cycle by some form of parallel activity such as Platform Development.*

Type: industrial project management metric

## Time-Cycle Constraint: Concept \*154. August 22, 2001tg 2333

In Evo delivery cycle planning, the maximum-recommended duration of a Frontroom delivery cycle (as opposed to a result cycle). The average duration time that a customer has to wait for a delivery.

Time-Cycle Policy: Average ‘customer trial delivery cycle’ is weekly on Monday morning.

Type: Evo term

## Time To Market: Concept \*365. August 22, 2001tg 2333

The calendar time between the [defined] start, or initial staffing, of the project, and project’s [defined level of] product successful release to [defined] market. A scale of measure.

Abbreviation: TTM

*Usage: This is essentially a measure of how fast an industrial enterprise can react from ‘opportunity or concept or contract’ (a defined kickoff event) to practical deployment of actual product. The shorter the time, the more competitive you are. Long TTM can lead to being too late to be a market leader, and to earn money on a market. Compare to Time Between Successive Products, which is the market’s view of the rate of putting product variants on a market.*

Type: Industrial product development metric

## Tolerable: Minimum Value Acknowledged: Concept \*539 June 4 2014, 2019

Tolerable *levels* of performance or resource use, are an acceptable range of a scalar attribute, that are better than failure levels, and worse than success (Goal/Budget) levels. *Tolerable* means: ‘in the tolerable range of attribute values’.

Tolerable *limits* are the *worst acceptable,* extreme edge-of-range, levels in the tolerable range. Tolerable levels are one type of scalar constraints, that intentionally constrain design choices.

Tolerable levels are any points in a tolerable range, of a scalar attribute, that are *outside* of the *intolerable* range (not failure or catastrophe range), and also outside of the *success* range of that attribute.

In simple analogy tolerable levels, the tolerable range, are temperatures that are neither too hot nor too cold. The temperature is tolerable, if not *optimal*, comfort.

Note may 20 2005 kai says delete and equate with catastrophe level

Synonyms [Tolerable \*539]:

• Acceptable \*539

Related Concepts [Tolerable \*539]:

• Fail \*098

• Limit \*606 (as in Tolerable Limit **(**539+\*606), which is implied when the parameter Tolerable is used alone)

• Range \*552 (as in Tolerable Range \*539 + \*552)

• Scalar Constraint \*468

• Tolerable Limit \*539 + \*606

• Minimum [Intel, 2003] \*539 + \*606

• Must Do [Historical Planguage Use] \*539 **+**606

Note:This replaced ‘Worst’ ( the abbreviation of Worst Case) in about 1989 when it became apparent that ‘Worst’, alone, as a parameter, was misunderstood as a target, not a constraint. A Douglas Aircraft engineer asked me, ‘Do you mean ‘must do’? ,and we changed it to that , then as a result. Looking back, my concepts of constraints and targets were not clearly articulated at that time. <-TSG

• Worst Acceptable Case \*539 + \*606

• Worst Case \*539 + \*606

Parameter [Tolerable \*539]: Tolerable. Or ‘Minimum Acceptable’. Or ‘Tolerable Level’, or ‘Tolerable Limit’.

Keyed Icon [Tolerable \*539]: >>

In context ----->>--->O------>>------>

Icon Background: it is based on a Goal symbol (>), and repeated multiple times for emphasis that it if you cannot reach the goal level, you should at least reach somewhere in the tolerable range (>>>>) and no less than the tolerable limit (>>).

Tolerable Range: >>>>>>>> in context: -------->>>>>>>>>----------->

----INTOLERABLE[TOLERABLE RANGE]>Goal------>

…………………..>>>>>>>>>>>>>>>>----------->

The tolerable range extends from the success level (the Goal level, ---->----) and then gets worse, to and including the tolerable limit (>>).

The keyed icons are

>> (tolerable limit icon)

or, >> >>>>>>> > ( the tolerable limit as edge of the tolerable range

or, for emphasis, **>>**>>>>>>>>**>**. The tolerable limit in bold (**>>**) and the goal level in bold (**>**).

Issue: November 24, 2004tg. What about upper and lower tolerable edges (too hot and too cold for comfort).

Type [Tolerable \*539]: Scalar Constraint

Note: term ‘tolerable’ suggested by Kai Gilb 29 May 2003

## Trade "trade" : Concept \*206. August 22, 2001tg 2335

A type of activity done by groups.

*For example environmental planning and regulation, international aid charity, mobile telephone products. E.g. ‘The computer trade’.*

## Translation [Planguage] "Translation" : Concept \*205. August 22, 2001tg 2336

A translation from (usually US English, the default language) a Planguage document to another major human language.

*Note: Planguage to UK English would be a dialect ‘tuning’ rather than a translation.*

## *Trend ?< Concept \*155 March 5, 2003*

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 [WELCH01, page 391]

Related Concept [Trend \*155]:

• Benchmark \*007

• Trend level (see Level \*337).

Keyed Icon [Trend \*155]: ?<

“Symbolizing a past (<) with some doubt (?) about the perfect truth.”

Must normally be applied on a scale icon (----?<----)

Type: Parameter, Benchmark.

Historical Note: This concept was suggested first by Kai Thomas Gilb, May 1995.

## True \*625 February 16, 2003

Predefined State. Same as On.

Related To:

• State \*174

## Type Concept \*398 May 5, 2003

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.

Examples:

Usability: Objective. “Implicit use of ‘Type.’”

Requirements Section. “Implicit type given by use in a heading.”

3. In this glossary, Type is used explicitly to state the categories of the Planguage concepts. A list of Planguage Types is given in Appendix TT.

[Editor Note to Publisher: This reference needs inputting.]

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.

Synonyms [Type \*398]:

• Concept Type \*398

*Type [Type \*398]: Parameter [Classification].*

## Ultimate. Concept \*442. August 22, 2001tg 2343

The systems engineering component, at the highest level worthy of consideration, is the ‘ultimate’ component.

Formal definition:

A system component which is ‘ultimate’ is one which is not considered worth exploring possible higher levels of conception.

Notes:

1. Ultimate is related to a defined stakeholder and a defined system.

What is ‘ultimate’ for one stakeholder or in one system or environment may not be the ‘end of the road’ for others with a different agenda.

2. It is not worth asking ‘Why?’ and finding a higher purpose or agenda for this component.

3. In traditional use of the term ‘Ultimate’ the emphasis is on the inability to go further. The point we make here is the unwillingness, or inefficiency of going further, for defined stakeholders and environments.

Usage:

Ultimate Objective

Ultimate values

Ultimate design

Source: Kai Gilb suggested this concept in connection with his Whirlwind manuscript July 22, 2001

## Ultimotive. Concept \*444 July 22, 2001 August 22, 2001tg

An ‘ultimotive’ is the highest level of motivation in a defined chain of motivations

A playful concept: concatenation of Ultimate Motivation

Usage: ‘an ultimotive requirement, leading to support for some funding’.

Source: concocted by Tom Gilb in conversation with Kai Gilb 22 July 2001.

## Uncertainty Concept \*310 November 27 2006

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.

Note: 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.

Related Concepts [Uncertainty \*310]:

• Risk \*309. 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 [BERNSTEIN96], 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.)

NOTE ADD A TABLE OF SMITHSON’S taxonomy OF different types of UNCERTAINTY ( SOURCE MEIJER COMPLEXITY PHD PAPER NOV 2006) Smithson, m ignorance and uncertainty, emerging paradigms, springerverlag new york 1989

Keyed Icon [Uncertainty \*310]: ±? “Deviation, ‘±’ and Doubt, ‘?’.”

Type [Uncertainty \*310]: Scalar Metric.

Domain [Uncertainty \*310]: Impact Estimation.

## Undeterminable: \*628 February 16, 2003

Predefined State. Attempt to determine has been made and the result is no clear state information is available.

Related To:

• State \*174

## Undetermined \*627 February 16, 2003

Predefined State. The state determination has either not been attempted or is not concluded yet.

Related To:

• State \*174

## Units Context: \*559 July 1, 2002 March 28, 2003

A units context defines where the scale of measure is applicable.

The units of measure ‘context’ can be rich in any useful dimensions and any useful information about the scalar level.

The units of measure context primarily contains:

• place: geographic location

• space: any other spatial notions, ‘where’

• event: any other conditions, particularly events or conditions

The units of measure context secondarily contains:

• sources: where does the specification come from

• authority: what power is behind the specification

• priority: any information helping establish the priority of the specification

• any other information useful or necessary for understanding the specification

The units of measure context can arbitrarily be specified in the following ways:

• in the Scale specification itself; explicitly and as a Scale Qualifier.

• in a benchmark, target or constraint specification; explicitly and as a Scale Variable.

(examples: ‘that are happy’ in Scale: % of Users that are happy)

Synonym [Units Context \*559]:

• Units of Measure Context.

## Units of Measure Concept \*560 May 3, 2003

Units of measure define the basic units to be used for measurement in a scale of measure.

Examples:

• Hours

• Kilometers per Hour

• Number of Days

• Percentage Probability

Related Concepts [Units of Measure \*560]:

• Scale \*132

Type [Units of Measure \*560]: Measurement Concept.

## Units Qualifier: \*561 July 1, 2002

Units Qualifiers specify in a Scale of measure which dimensions apply to the units of measure.

Examples: per year, per person.

For example the ‘per year’ in

Scale” % Past Year Customers who revisit shops .

‘%’ is Unit of Measure (\*560).

‘Past Year Customers who revisit shops’ is Units Context (\*559).

## Unknown \*629 February 16, 2003

Predefined State. The state is not known.

Related To:

• State \*174

## Unstable \*630 February 16, 2003

Predefined State. The state is changing in unpredictable ways.

Related To:

• State \*174

## Until \*551 March 5, 2003

‘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).

Examples:

Fail [GB, Next Version]: 60% Until Survival.

Survival [International, Next Year]: 20% Until 0%.

Fail [EU]: 80% -> 20%.

Related Concepts [Until \*551]:

• Or Worse \*549

• Or Better \*550

• Range \*552

Keyed Icon [Until \*551]: -> “A right arrow between *two scalar levels* indicates a range.”

Note: the ‘->’ icon is also used for Impacts (A -> B) but in the context of a numeric range (22 -> 66) it cannot be misunderstood.

Type [Until \*551]: Logical Operator.

## Upstream Concept \*291 November 21, 2002

‘Upstream’ refers to any earlier processes than the one currently identified (the current reference point).

Example:

Requirements is upstream of its consequent design, and the design is upstream to its implementation. Design is downstream to its requirements

Parameter: Upstream, synonym or long form = Upstream From.

Keyed Icon: ~> A ~> B means A is upstream of B

Note: similar to Impacts symbol ->, Upstream impacts downstream. Flow is from upstream to downstream (of water and defects!).

Example:

Fail [Test Processes ~> Customer Acceptance Test]: 99%. “Means the Fail limit for all test processes upstream of Customer Acceptance Test is 99%.”

Related Concept: Downstream (\*052)

Type: Parameter [Relationship].

Domain: Process.

## User Concept \*234 May 6, 2003

A user is a person or group, who actually will make *practical* use of a system.

Related Concepts [User \*234]:

• Stakeholder \*233

• Client \*235

• Consumer \*038

Type [User \*234]: Role [Stakeholder].

## User-Defined Term Concept \*530 March 4, 2003 B tg

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,’ 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, and may be specified using ‘Defined’ or ‘Defined As’, or by adding a tag to any expression, statement, or term.

Examples:

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).

Related Concepts [User-Defined Term \*530]:

• Planguage Term \*211

• Project Language \*247

Synonyms [User-Defined Term \*530]:

• Planguage-User Defined Term \*530

Type [User-Defined Term \*530]: Planguage Concept.

## Validation Concept \*601 October 24 2004

Validation is a process of confirming that a specification is complete and correct.

Notes:

1. There are many methods of validation including, review, role playing sessions, simulations, prototypes, and feedback from Evo steps.

Synonyms [Validation \*601]:

• Specification Validation

Related Concepts [Validation \*601]:

• Specification Quality Control (SQC) \*051.

• Specification Content Review (SCR) \*542.

• Verification \*321

Type [Validation \*601]: Process [Development].

Note : 14 August 2006, maybe we should have a Planguage parameter called; ‘Validated’ to specify exactly who or how a particular detail {Scale, Meter, Goal, Past, Fail, etc.) of a larger specification was validated.

## Value (Stakeholder-Value) Concept \*269 March 12 2005, 14 May 2013, 19 June 2013 (will or do, note 4 & 5 & 7)

Value is *perceived* benefit: that is, the benefit we *think* we, will or do, 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 and is objectively measurable, independent of actual stakeholder perceptions (value).

5. Value is relative to a defined stakeholder, under defined conditions [Qualifiers]: it is not absolute.

6. 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.

7. There are many Planguage ways of indicating that a stakeholder values an attribute. These include using Stakeholder, Authority, Impacts, Qualifiers, and Source parameters.

8. Value perception can related to any time, past, present or future.

“Nowadays, people know the cost of everything   
  
and the value of nothing.”

Oscar Wilde.

**

Oscar Wilde. Irish Poet, wit and dramatist (1854 – 1900)

Keyed Icon [Value \*269]: +> ‘adds value to’

Explanation: A +> B means A (some action) adds value to B.

Reliability:

Scale: MTBF (Mean Time Between Failure).

Goal [Deadline = 1st Release, Area = EU, Population = Teenage Users] 30,000 hours.

Value [Goal, At Release] +> Early Sales +> Market Share +> Annual Revenue.

A value chain: The benefit of reaching the specified Goal is indicated as a series of added value impacts (‘+>’) to different performance indicators. We value the 30,000 hours MTBF (mean time between failure) for the ultimate benefit we expect to get from Annual Revenue/

Synonyms [Value \*269]:

Worth \*269

Stakeholder Value \*269

Related Concepts [Value \*269]:

• Benefit \*009

• Impacts \*334

• Values \*592

• Quality \*125

Type [Value \*269]: System Concept, Priority Data.

## Value Chain \*591 May 24, 2003

A value chain is a sequence of two or more defined objects. The initial object produces value, as defined by the second object’s value. This value in turn may produce value in a next object, and so on, as long as we care to define the chain of value.

The exact nature of the value and how it is produced can be specified in more detail in an object specification (such as a design template).

Example:

Value: A +> B +> C. “This means A impacts the value of B, which in turn, impacts the value of C.”

Notes:

1. Value is produced for defined stakeholders, in defined places at defined times under defined conditions. This information can be given with qualifiers in a value chain. See the Value concept.

Related Concepts [Value Chain \*591]:

• Value \*269

• Impacts \*334, and Impact \*087

• Supports \*415

Keyed Icon [Value Chain \*591]: ‘+>’

Type [Value Chain \*591]: Specification Element.

## Values Concept \*592 May 4, 2003, June 19 2013 List

Values are human culture guidelines that help individuals or groups make choices. They are the principles, standards and paradigms that we consider when choosing alternatives.

Values include ethics, religious, professional and legal notions that we choose to let us be guided in decision making.

*“What are values? Values are principles used for evaluation. We use them to evaluate the actual or potential consequences of action and inaction, of proposed alternatives, and of decisions. They range from ethical principles that must be upheld, to guidelines for preferences among choices.*

*We can identify our values by hard thinking. We make them explicit through statements expressing value judgments. To render value judgments useful in decision making, we must be precise about their meaning. We can articulate this meaning qualitatively by stating objectives and, if desirable, we can embellish it with quantitative value judgments.”*

Ralph Keeney in ‘Value-Focused Thinking’ [KEENEY92, pages 6 and 7]

A List of 418 Human Values is (2013) at

<http://www.stevepavlina.com/articles/list-of-values.htm>

and there are other internet lists.

Related Concepts [Values \*592]:

• Value \*269: Value is something inherent in a system.

Type [Values \*592]: Guidelines.

## Value To Cost Ratio Concept \*635 May 26, 2003 B

The value to cost ratio is a numeric expression of perceived stakeholder value to estimated or actual costs, for a set of design ideas or for an Evo step.

Notes:

1. It is useful for making comparisons of design alternatives and proposed Evo step alternatives, in order to pick the one with the best value to cost ratio.

2. Bear in mind that for a given set of design ideas or Evo step, the value to cost ratio will differ from stakeholder to stakeholder.

3. Value to cost *scores* can be used where high-level discussion is appropriate, and detail is not available. Maybe as a result, the best options can be identified for further investigation to estimate their value to cost *ratios*.

4. ‘Value to cost ratio’ must be distinguished from ‘performance to cost ratio.’ Performance to cost ratio is an estimate/measure of the percentage performance improvement, which will be delivered/is delivered compared to cost.

Related Concepts [Value To Cost Ratio \*635]:

• Value \*269

• Cost \*033

• Value to Cost Score \*349

• Performance to Cost Ratio \*010

Type [Value To Cost Ratio \*635]: Design Analysis Metric.

## Value to Cost Score Concept \*349 May 26, 2003 B

The value to cost score is a simple numeric scoring mechanism, sometimes used in meetings to share opinions, and to arrive a consensus about the value and cost of Evo steps. The scale is 0 to 9, where 9 is the score representing highest value, or highest cost. The first number is the Value estimate, and the second is the Cost estimate.

Example:

*Step 1: Modify Old System with New Computer Hardware* 🡪 *6/1 Value-to-Cost Score.*

*Step 2: Optimize Software* 🡪 *7/3.*

*Step 3: Reduce Sales Price* 🡪 *5/3.*

Three Evolutionary Steps with the ‘🡪’ impacts symbol used to indicate that the particular step ‘impacts’ by the specified ratio. We do not say exactly what it impacts, since the ratio is general, subjective, and covers all values and costs (which are not actually identified).

Notes:

1. An example of “9/1” for an Evo step would indicate high value for the step and very low cost. The ratio can be used to prioritize and sequence Evo steps.

2. This method cannot replace more-detailed estimations, based on quantified requirements and Impact Estimation. But, it is useful in meetings and for discussion purposes, when the rigor of multiple requirement quantification would be unnecessarily formal.

Type [Value to Cost Score \*349]: Design Analysis Metric.

## Variability: Concept\*582 July 13, 2002

Any development artifacts or components that do not have identical corresponding artifacts in another product, represent variability between the products.

The is a Product Line related concept.

See Commonality

Thanks to Stefan Ferber, Bosch, for suggesting this concept. July 2002.

## Variant: Concept \*203. *August 24, 2001 tg* 2150

A special variation of a Planguage artifact *{handbook, slides, etc.}.*

*For example the Variant could be made for a company (like Nokia), or for a Trade (like Software Engineering).*

## Verification Concept \*321 May 9, 2003 B

A verification process tries to confirm that the output of a process corresponds to expectations. Verification can use a variety of means. Verification needs to know the process inputs, and the rules or requirements that apply to that process, in order to determine expected outputs with which to compare reality.

Notes:

1. Any applicable verification processes may be used, including Specification Quality Control (SQC), Evolutionary feedback evaluation, PDSA Study processes, testing, and user feedback.

2. Verification has 4 major elements[[19]](#footnote-19):

* Inspection: Confirmation that the product has met its requirements by visual examination. For example: Looking at a coding listing to verify that the Naming Convention requirements have been implemented correctly.
* Analysis: Confirmation that the product has met its requirements by performing an analysis of data about the product to pass/fail criteria. For example: Meeting a bridge capacity weight requirement by mathematically determining how much weight the bridge can hold without having to load it until it collapses.
* Demonstration: Confirmation that the product has met its requirements by showing functionality in a manner that does not have strict qualitative pass/fail criteria. For example: Demonstrating a user interface concept without meeting specific timing or human factors requirements.
* Test (Formal): Confirmation that the product has met its requirements through technically rigorous testing to specific pass/fail criteria. In other words, manipulation of the product to determine how it responds when compared quantified success data points. For example: Formal "testing" that requires such things as test scripts, documented test results, and accompanying test reports to show that the requirements are met.

Related Concepts [Verification \*321]:

• Validation \*601

• Specification Quality Control (SQC) \*051

• Evolutionary Project Management \*355

• Study [PDSA] \*171

Type [Verification \*321]: Process.

## Version Concept \*332 February 27, 2003 B

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 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’.

Examples:

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.

Examples:

Usability: 9-Jan-03. Scale: Time to <learn>. Goal: 6 hours or less.

Or (more explicitly)

Usability: Version = 9-Jan-03. Scale: Time to <learn>. Goal: 6 hours or less.

Usability: [Version = 9-Jan-03] Scale: Time to <learn>. Goal: 6 hours or less.

Synonyms [Version \*332]:

• Edition \*332

• Instance \*332

Type [Version \*332]: Parameter [Specification Control].

## Vision Concept \*422 February 21, 2003

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***[[20]](#footnote-20)***

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. 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.

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’.

Type [Vision \*422]: High-Level Requirement.

Related Concepts [Vision \*422]

• Core Ideology "controllable" . Concept \*664

• Story. Concept \*655 . See Stephen Denning.

## Veto Constraint. Concept \*462. December 21, 2001

A veto is a negative constraint; the type ‘Don’t do X’.

Type: non-scalar constraint. Requirement

Domain: CE.Planguage.requirements.constraints. constraints.veto.

Synonyms: ‘Do Not’ (Constraint), prohibition, Taboo, negative non-scalar (binary) constraint, veto

Related Concepts:

• Demand constraint, \*461

Examples [veto constraint]:

P1: Constraint:Products and services of direction competitors shall be avoided.

P2: Constraint:No software product shall be released for sale until at least 3 month field trial is conducted for the version to be released. <- Technical Director’s Policy 6.9.

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

## View Concept \*484 January 9, 2004

A view is any useful perspective regarding a defined system.

3.9 view: A representation of a whole system from the perspective of a related set of concerns.

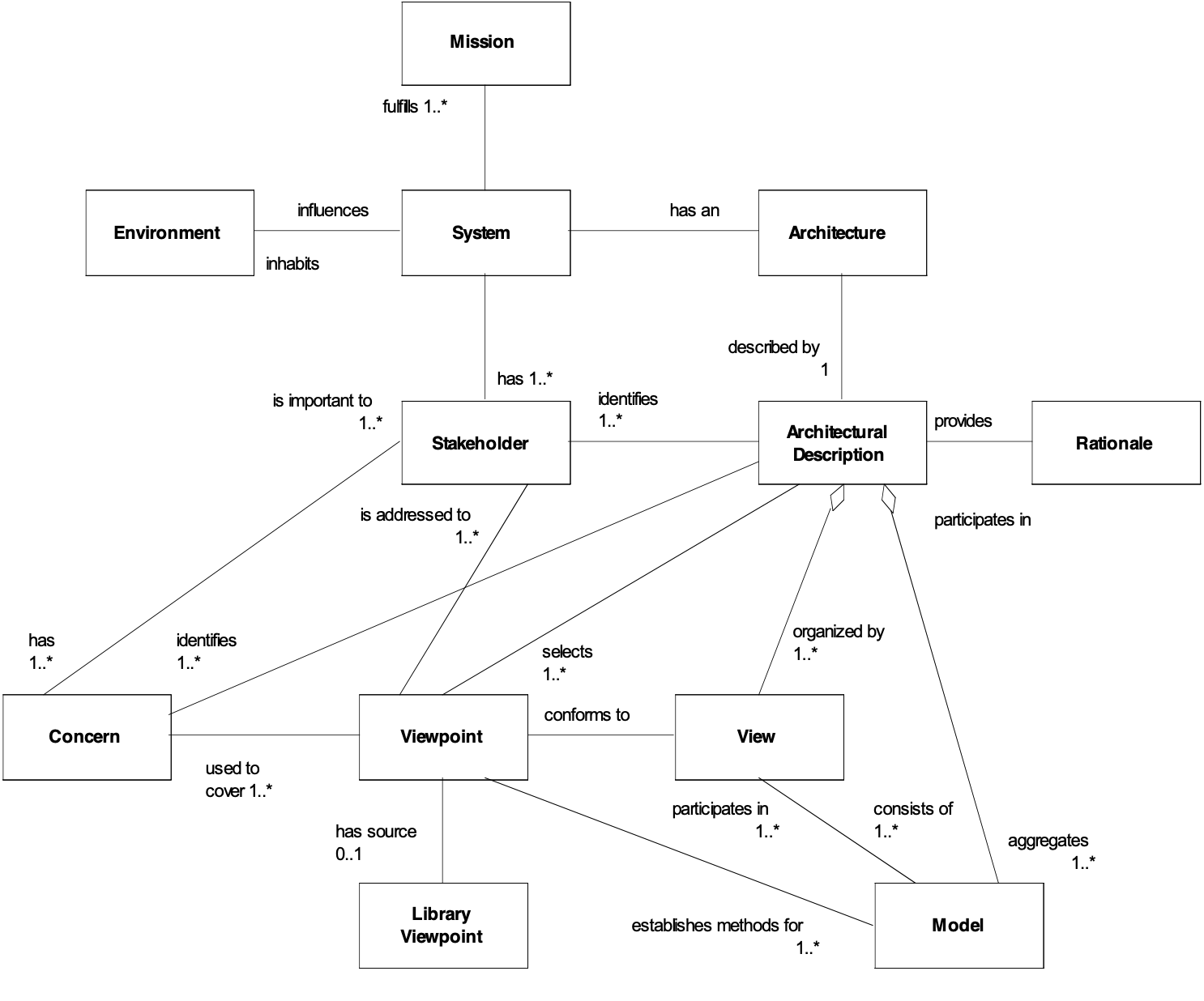
IEEE 1471 – 2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems

Notes:

1. A view is defined for a purpose or has ‘direction’.

2. A view allows us to focus our attention temporarily or regularly in a specialized manner.

3. A view is ‘what we see’, a viewpoint is where you look from (M. W. Maier, *. IEEE Architecture Working Group, used with permission of the author.).*

Figure: The IEEE 1471 Standard conceptual framework. From Aerospace Corporation, “*The IEEE 1471-2000 Standard - Architecture Views and Viewpoints” presentation by Mark W. Maier et al. IEEE Architecture Working Group. Used with permission of the author.*

Examples:

View: Country = {USA, Canada}, System = {Military Communication Systems}, Time = 2000 and Later.

The Reliability Perspective: View: Country = {USA, Canada, UK}, System = {Military Transportation Systems}, Time = 2003 and Later, Performance = Reliability.

Synonyms [View \*484]:

• Perspective

Related Concepts [View \*484]:

• Scope \*419

• Viewpoint \*644

• Description \*416

• Specification \* 137

• Subjectivity: \*387

Type [View \*484]: System Concept.

## Viewpoint Concept \*644 January 1, 2004

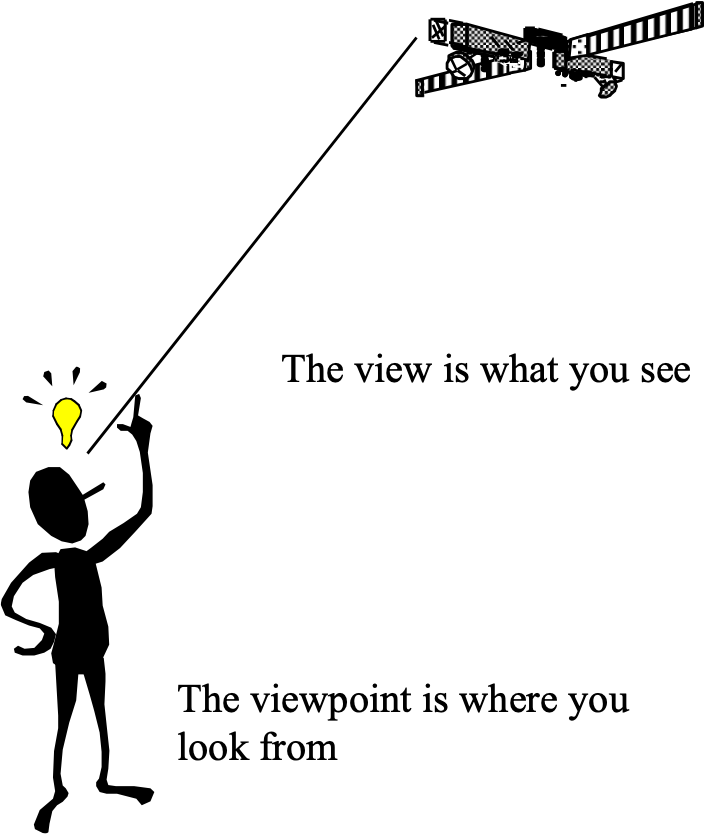
A viewpoint captures the rules for constructing and analyzing a particular kind of view. It is a template for a view and can be reused across many architecture descriptions.

Figure: The View – Viewpoint distinction. Source Aerospace Corporation, “*The IEEE 1471-2000 Standard - Architecture Views and Viewpoints” presentation by Mark W. Maier et al. IEEE Architecture Working Group (slides).*

Notes:

1. “A view belongs to a particular architectural description. A viewpoint captures the rules for constructing and analyzing a particular kind of view. It is a template for a view and can be reused across many architecture descriptions. View type was considered as an alternative term for viewpoint because of the strong analogy of view and viewpoint to instance and type; but we chose "viewpoint" because of its use in existing standards and the requirements analysis literature. Separating the two concepts is important for writing standards, although not essential if one’s scope is simply the description of a single system. “

Source: Introducing IEEE Standard 1471: Recommended Practice for Architectural Description for Software Intensive Systems Mark W. Maier, The Aerospace Corporation David Emery, MITRE Rich Hilliard, ConsentCache, Inc., part 6.

1. “The key idea is that a viewpoint identifies a set of concerns about an architecture, and how the architectural description will address these concerns, in terms of languages and notations, models and analytical techniques, methods, etc.”

Source: Introducing IEEE Standard 1471op. Cit.., part 7.

3. Planguage is one of many possible basic resources for constructing a viewpoint. Planguage templates, for requirements or design, are specific examples of practical viewpoints.

Related Concepts [Viewpoint \*644]

• View \*484

• Template \*254

• Form \*068

• Standard \*138

• Rule \*129

• Stakeholder \*233

• Impact Estimation Table \*638

Type [Viewpoint \*644]: Role

## Waterfall Method Concept \*311 February 25, 2003

The analogy in the term, ‘Waterfall Method’ refers to a series of waterfall stages as the ‘water’ (project progress) flows downstream. The basic concept of the Waterfall Method is that projects will define and freeze approved requirements, then define and freeze approved design to meet those requirements, then build the design, and then carry out acceptance tests to test that the requirements are met. The entire system is in principle, delivered in one ‘big bang’ to the client.

Notes:

1. A project management lifecycle method, which was popular amongst software engineers in the 1980’s.

2. The Waterfall Method contrasts unfavorably with the Evolutionary Project Management method as it lacks feedback and learning mechanisms. There is no ability to adapt as more information becomes available, and as the external business environment changes. Importantly, there is no possibility of early delivery of any business benefit.

Related Concepts [Waterfall Method \*311]:

• Evolutionary Project Management (Evo) \*355

Type [Waterfall Method \*311]: Method.

## *Wish >? Concept \*244 October 9, 2004*

A Wish parameter is used to specify a stakeholder-valued, not-committed target level for a scalar attribute.

A Wish level is specified on a defined Scale, under specified conditions [time, place, event, scale variables].

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, then Wish.)

4. A ‘Wish’ specification can apply to a performance or resource requirement.

5. Conditions for Conversion from Wish to Goal

1 Feasible (technically): the Wish level much be within the state of the art.

2. Profitable: the long term benefits of delivering the Wish level must exceed the life cycle costs.

3. Resource Available: the necessary resources to deliver the Wish level must in fact be available.

4. Prioritized: even if the Wish level is feasible, profitable, and there exist enough resources; it must be prioritized in front of other desired performance levels that meet these conditions. This generally implies that this Wish level needs to be the most profitable option in order to get converted to the more committed priority target level, the Goal level.

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.

Synonyms [Wish \*244]:

• Wish Target \*244

• Wish Level \*244 (see Level \*337)

Related Concepts [Wish \*244]:

• Need \*599

• Target \*048

• Goal \*109

• Budget \*480

• Stretch \*404

• Ideal \*328

Keyed Icon [Wish \*244]: >? “A perhaps questionable goal or budget.”

In context: ---->?--->O--->?--->

Type: Parameter, Target.

Historical Note: The Wish parameter was first suggested in December 1995 by the Scottish Widows organization, through Dorothy Graham of Grove Consultants.

## Work-Hours: Concept \*156 Not edited yet.

The total hours of work, irrespective of clock-hours, or number of people involved. The cost of doing something.

*The older term ‘man-hours’ is not ‘politically correct’; and in any case is a bad description of the hours needed to do work! The prefix ‘work-‘ can obviously be used with other units of time; work-minutes, work-months, work-week, work-year, engineering work-years, work-weeks [{engineers, management, support staff}].*

## Workload. Concept \*459. August 10, 2001 1615

The capacity for a defined type of work.

More formal definition template:

An amount of work to be done in a specified [Time] by a specified [Agent] on a specified [System] in a specified [Environment].

Type: System attribute.

Domain: System.Attributes.Performance.Workload.

Related Concepts:

• Performance \*434: the basic umbrella concept for system effectiveness including {Quantity, Quality, Savings.}

• Quantity \*437: the set of conventionally quantified performance concepts including Savings and Workload attributes.

• Savings \*429: degrees of resource savings in this system compared to a reference system

• Quality \*125: measures of how well the system performs, as opposed to how much (Quantities).

Synonym: Work Capacity \*459, Workload Capability, Workload Capacity.

## Workload Capacity Concept \*459 December 5, 2004

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 captures *benchmark* information. It expresses 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 can be used to express 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 [Workload Capacity \*459]:

• Work Capacity \*459

• Workload Capability \*459

• Workload \*459

• Capacity \*459

Related Concepts [Workload Capacity \*459]:

• Performance \*434

• Quality \*125

• Resource Saving \*429

• Workload Capacity Requirement \*544

Type [Workload Capacity \*459]: Performance Type.

## Workload Capacity Requirement Concept \*544 March 24, 2003

A workload capacity requirement is a need to achieve specific work capability levels for a system: ‘how much’ work a system should do.

Synonyms [Workload Capacity Requirement \*544]:

• Work Capacity Requirement \*544

• Workload Capability Requirement \*544

• Workload Requirement \*544

• Capacity Requirement \*544

Related Concepts [Workload Capacity Requirement \*544]:

• Performance Requirement (Objective) \*100

• Workload Capacity \*459

• Workload Capacity Requirement Specification \*544 & \*137

Type [Workload Capacity Requirement \*544]: Performance Requirement Type.

## Work process "work process" .. Concept \*182. August 24, 2001tg 2315 "\*182.work process" .

A defined process for carrying out human work.

*Procedures and Rules give examples of formally defined processes for doing work. See the book index for concrete examples. Most of this book defines one way or another a work process for this method.*

Type: enigneering.process

## Work Process Standard "standard" : Concept \*138. August 27, 2002

A work process standard is an ‘official’ written specification that guides a defined group of people in doing a process. It is a best-known practice.

Synonym: Standard \*138 *(implied ‘for work processes’)*.

*Standards include: {process (\*113) "processes" ,* ***rule (\*****129) "rules" , procedure (\*115) "procedures" , entry condition (\*056) "entry conditions" , exit condition (\*064) "exit conditions" , form (\*068) "forms" , rate (\*139) "rates" , policy (\*111) "policies" , checklist (\*016) "checklists" and defined terminology (defined \*045, Clarity Rules \*129, Content Rules \*543.*

*Related Concepts:*

*• Template, \*254:* An example or ‘model’ of something, which can be used to help people to tailor or make something based on that model. A template can be for something other than a ‘standard’. A generic standard can be presented in a form other than a template (for example a form or list or model).

*• Generic Standard, \*150:* A 'generic' standard is an example, model, specification or template of a work process standard, which can be tailored by the reader to suit their purposes.

Type: engineering.process.standard

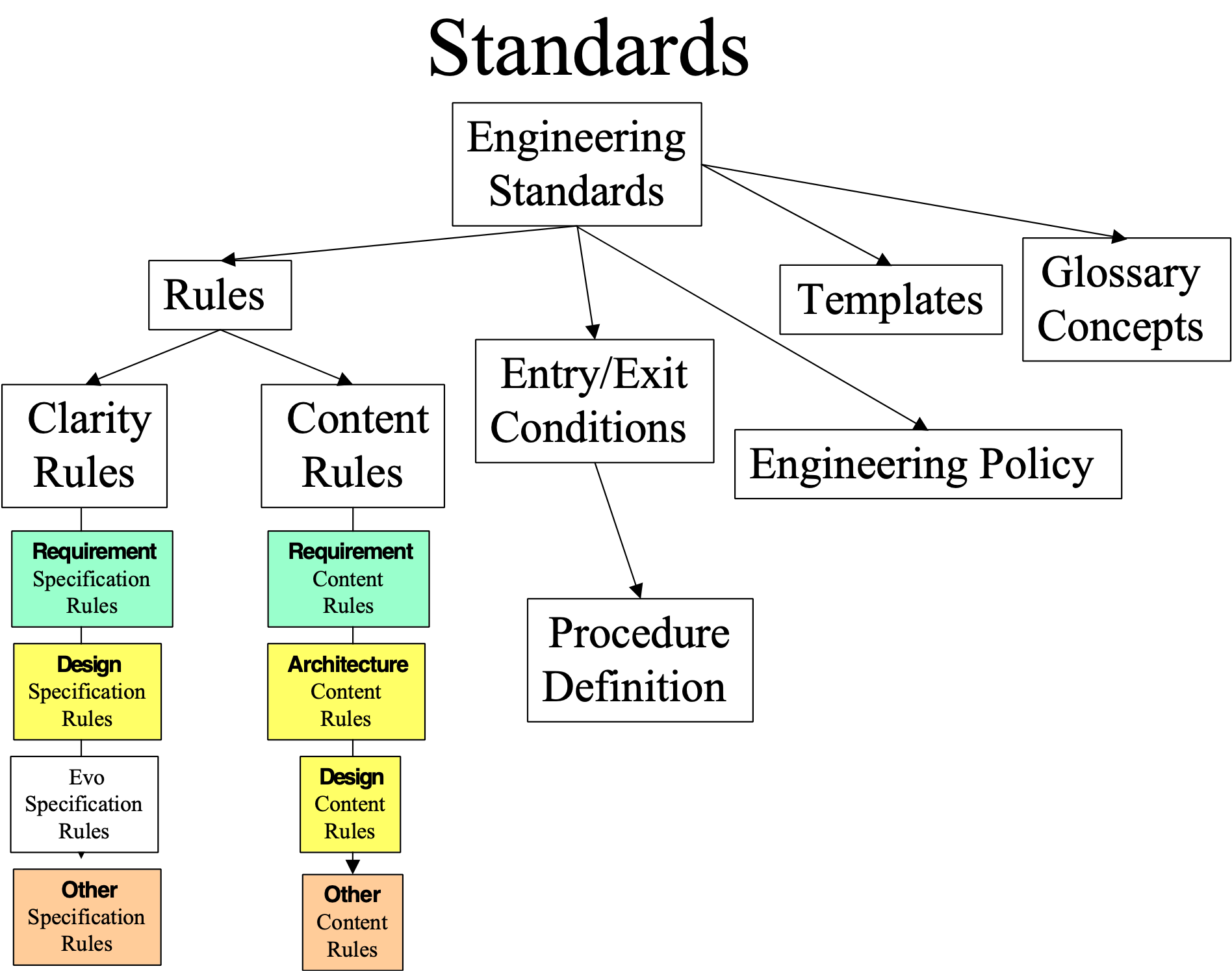


Fig. \*138 There are a variety of work process standards provided by Planguage to help define work processes.

## Work Rates "work rates" : Concept \*139. August 24, 2001tg 2331 "\*139.work rates" .

The speed with which a defined work task is done.

*See optimum checking rate. Synonym: rates.*

## Worst Acceptable Case Concept \*539 October 30, 2004

See Tolerable (Level) Concept \*539

## Writer: Concept \*004. August 24, 2001tg

## See Author.

## XREF: Concept \*258. August 24, 2001tg

## A Planguage-defined abbreviation meaning ‘cross reference’.

1. 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! [↑](#footnote-ref-1)
2. Thanks to Chris Dale for pointing out the necessity of including ‘*in combination with other factors Feb 2007*’ [↑](#footnote-ref-2)
3. *With thanks to Don Mills, NZ* [↑](#footnote-ref-3)
4. [↑](#footnote-ref-4)
5. Because these stakeholders can only receive benefit from an operational system, they can be referred to as “operational stakeholders”. [↑](#footnote-ref-5)
6. Or “Originators” [↑](#footnote-ref-6)
7. This is not the same as Context (\*483). None of these terms is entirely satisfactory, but “context” comes closest to the idea of a simple “frame of reference” or environment within which action may take place (including the operation of a system). [↑](#footnote-ref-7)
8. “Discretion” might best be described here as “implicit authority”. Implicit or explicit authority could be regarded as the result of “stakeholder prioritization”. [↑](#footnote-ref-8)
9. Ideally, a requirement should have a single Sponsor. A group may be collectively responsible, but even then, it is preferable for one among them to take on the role of Sponsor. [↑](#footnote-ref-9)
10. “…for the time being” could be added, authority always being conditional. If the conditions are not met, the authority is no longer valid and, as an immediate and automatic consequence, the “requirement” reverts to being a “potential requirement”. [↑](#footnote-ref-10)
11. Strictly, either a sub-system or an attribute (\*003) of the system (\*145). [↑](#footnote-ref-11)
12. We can distinguish here between transitional context (e.g. a project) and operational context (where “environment” is a common alternative term). If the transitional context is undefined, we can refer to the requirement as an “orphan requirement”. If the operational context is undefined, we can only say that the Requirement Specification (\*508) is defective; the status of the requirement itself is undecidable (but decidedly not authorized). [↑](#footnote-ref-12)
13. [↑](#footnote-ref-13)
14. @BOOKLET{www:RequirementsCapturingByTheSystemArchitect,

    AUTHOR=" Muller, Gerrit",

    TITLE=" Requirements Capturing by the System Architect",

    HOWPUBLISHED="\url{http://www.extra.research.philips.com/natlab/sysarch/RequirementsPaper.pdf}",

    YEAR=" 1999",

    } [↑](#footnote-ref-14)
15. *The New Shorter Oxford English Dictionary, 1993. Oxford University Press.*

    *ISBN 0-19-861134-X.* [↑](#footnote-ref-15)
16. Contributed by Niels Malotaux. [↑](#footnote-ref-16)
17. Contributed by Niels Malotaux. [↑](#footnote-ref-17)
18. Cited page 306 Principles of Software Engineering Management ( so permission obtained for that) [↑](#footnote-ref-18)
19. *With thanks to Lawrence Day, Boeing.* [↑](#footnote-ref-19)
20. FROM: http://www.ku.edu/carrie/docs/texts/mlkdream.html

    Martin Luther King, "I Have a Dream" Speech

    August 28, 1963, Washington D.C. [↑](#footnote-ref-20)