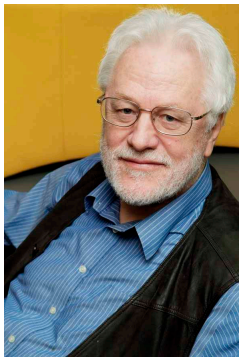
Competitive Systems Engineering

How to do systems engineering in hot competition.
Detailed pragmatic and unconventional techniques.

Tutorial
June 2008 INCOSE Symposium, Utrecht Holland
One Day
Instructor: **Tom Gilb**
Result Planning Limited



"Competitive" Engineering?

Competitive Engineering

- Keeps the engineers focus on
  - **Winning**
  - **Beating** Competition
  - **Improving** your competitive position
  - Making your product or system **the best**
  - Looking at the **future of competition**
    - Not just what it
    - But, what **will** be

Engineering

- Design to Specifications
- Even if specifications are
  - 'uncompetitive'

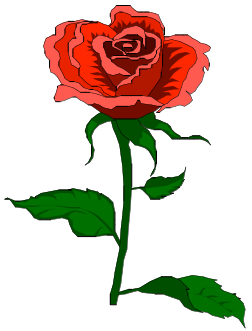
Detailed Tutorial Outline:
The Competitive *Tools*

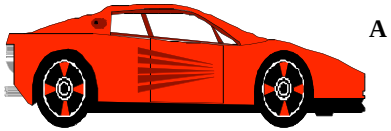
- **Planguage**: a quantified planning language.
- **Integrating** benchmarks and requirement targets
- Quantified **Quality Control** of specifications
- **Impact Estimation** Tables for quantified evaluation of design
- **Evolutionary** Project Management

Consider the Performance of :

A flower

- **fragrance**
- **attractiveness**
- **pollen quantity**
- **toxicity**
- **bloom frequency**



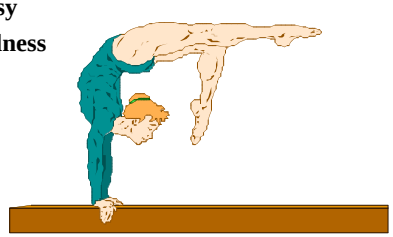


A car

- **comfort**
- **safety**
- **speed**
- **capacity**

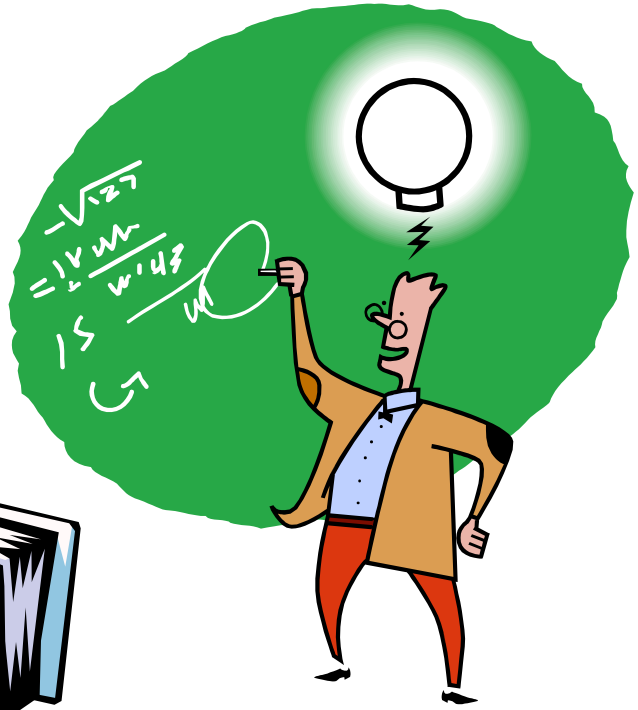
A person

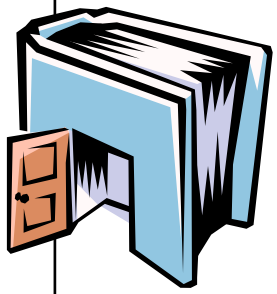
- **balance**
- **intelligence**
- **courtesy**
- **helpfulness**



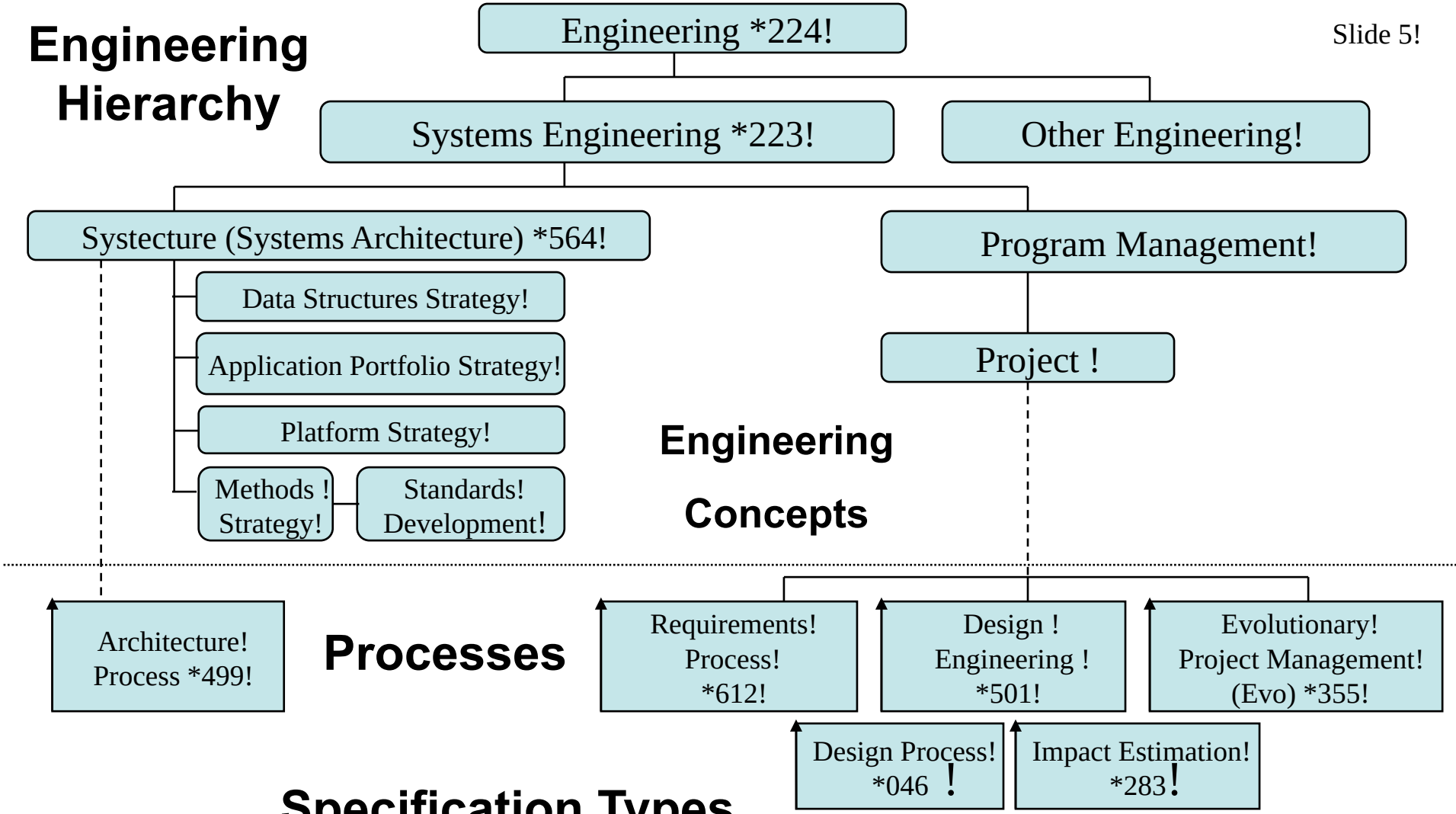
Tutorial Objectives: Insight into Competitive Tools

- 1. Become aware of entirely **new ideas**.
- 2. Be able to **evaluate** if these apply to participant's work.
- 3. Be aware of how to **get more** detailed information on the subjects.
- 4. **Enthuse** participants with the attractiveness of the ideas presented.

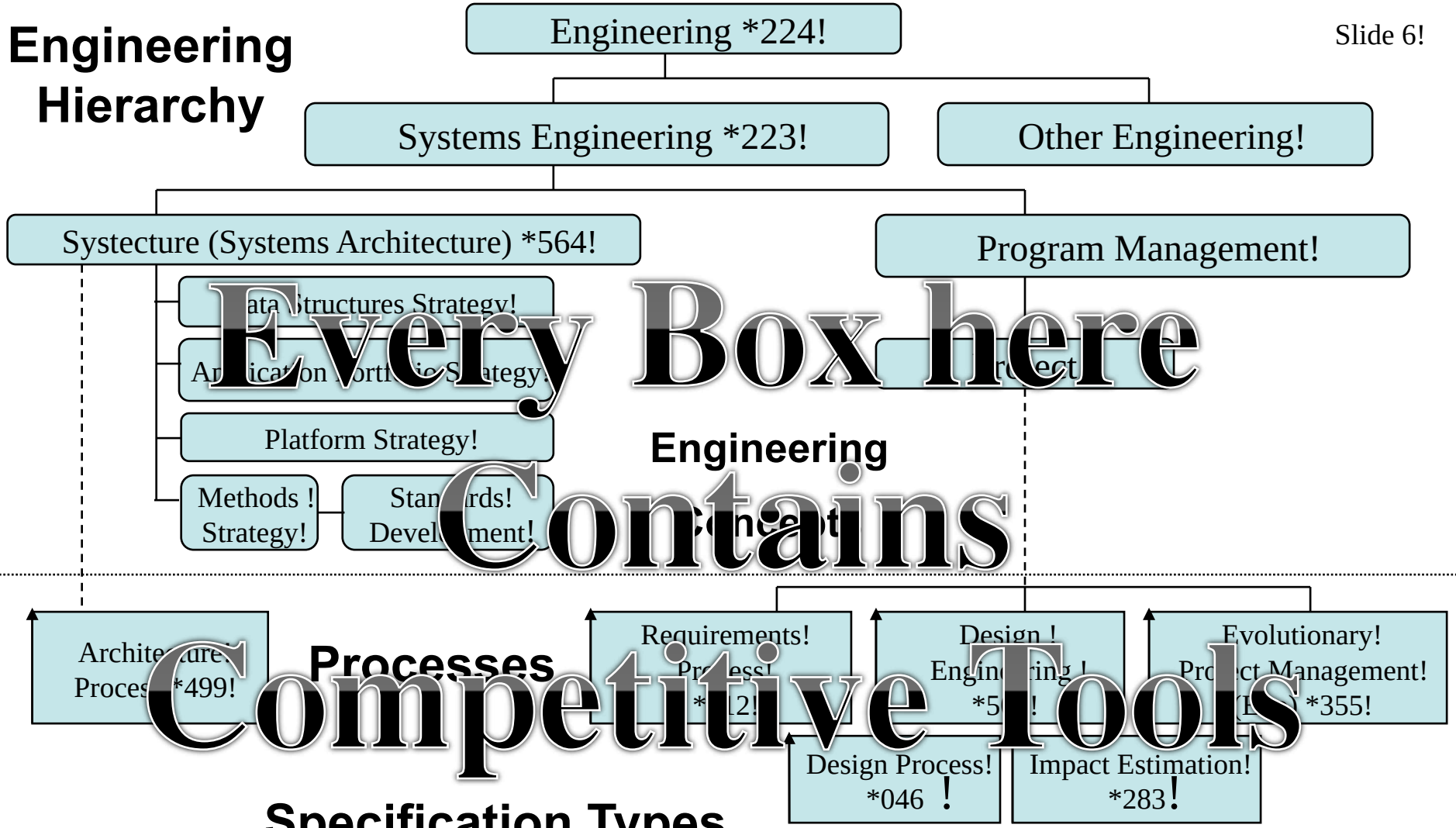




Engineering Hierarchy



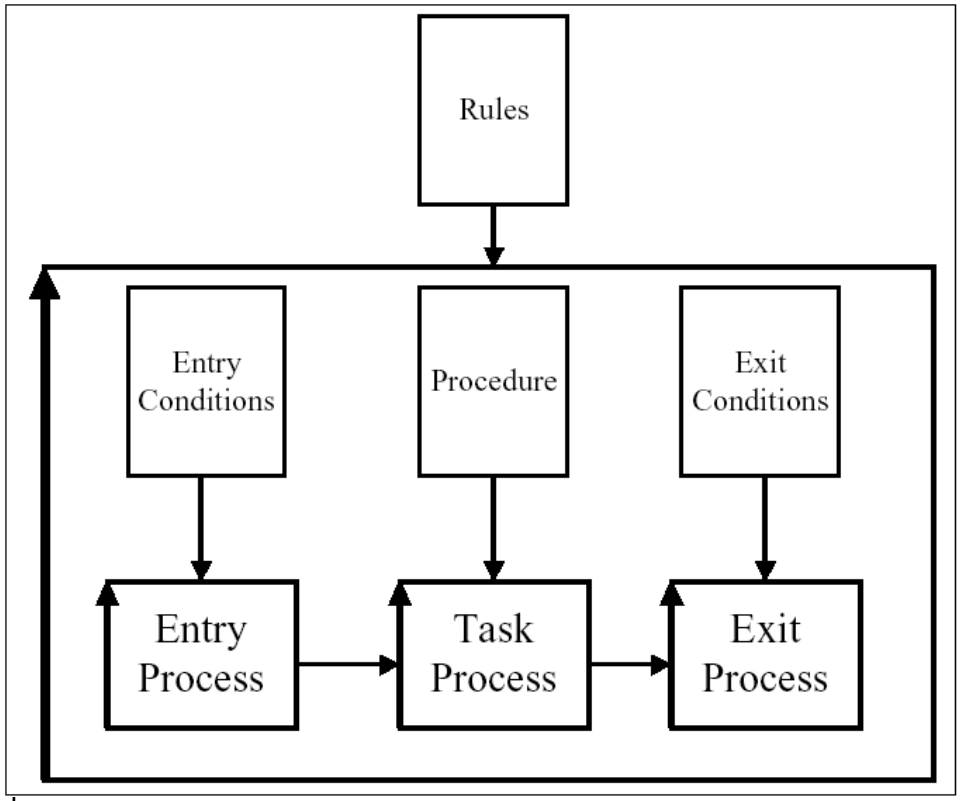
Engineering Hierarchy



Every Box here Contains Competitive Tools

Part 1: Planguage: a COMPETITIVE quantified planning language.

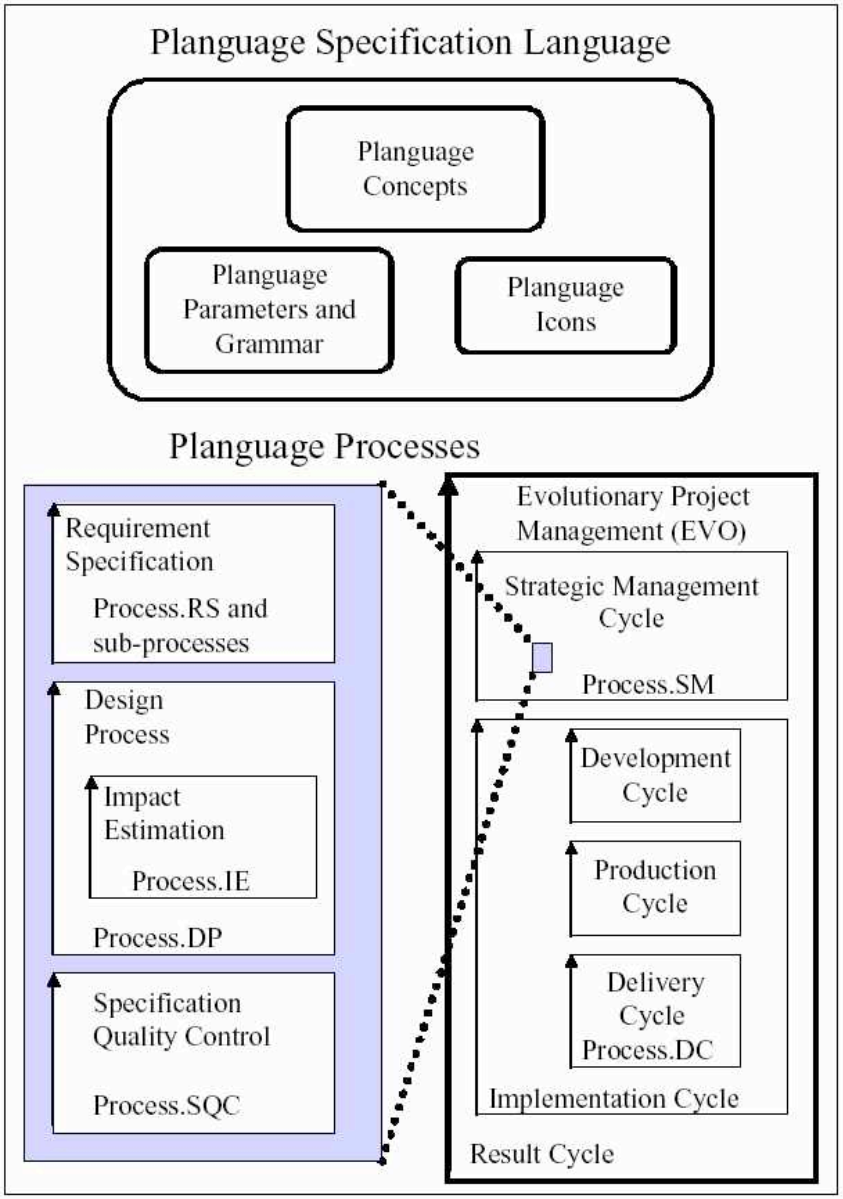
- A Planning Language - an engineering language
- A systems engineering language (software, management)
- Concept Glossary
- Graphical Language
- Control of Multiple dimensions: Performance, Costs, Constraints
- Extendible, Tailorable, Open
- Rich views, traceability, configuration management
- Risk Management
- Priority Management



A Planning Language - an engineering language

Used for

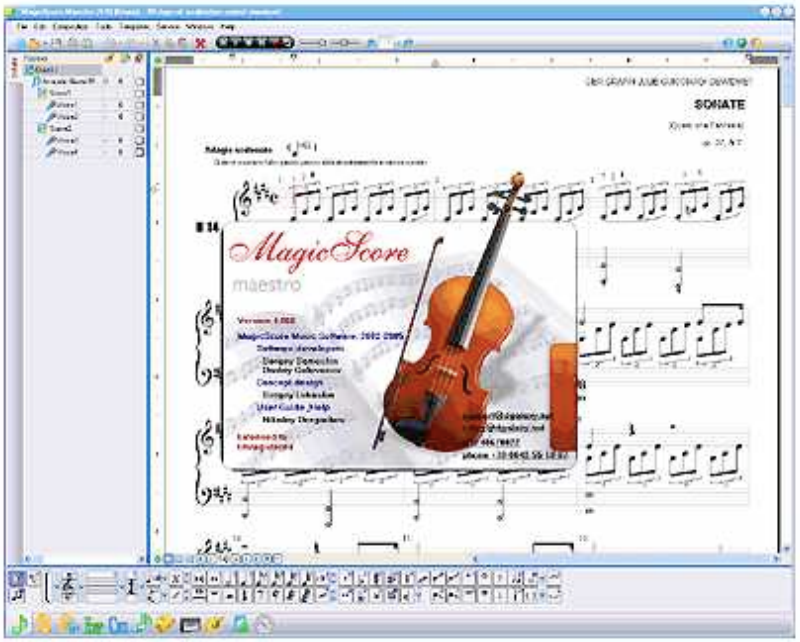
- Systems Analysis
- Requirements
- Contracting specs
- Design - Architecture
- Presentation
- Spec Quality Control
- Project Management

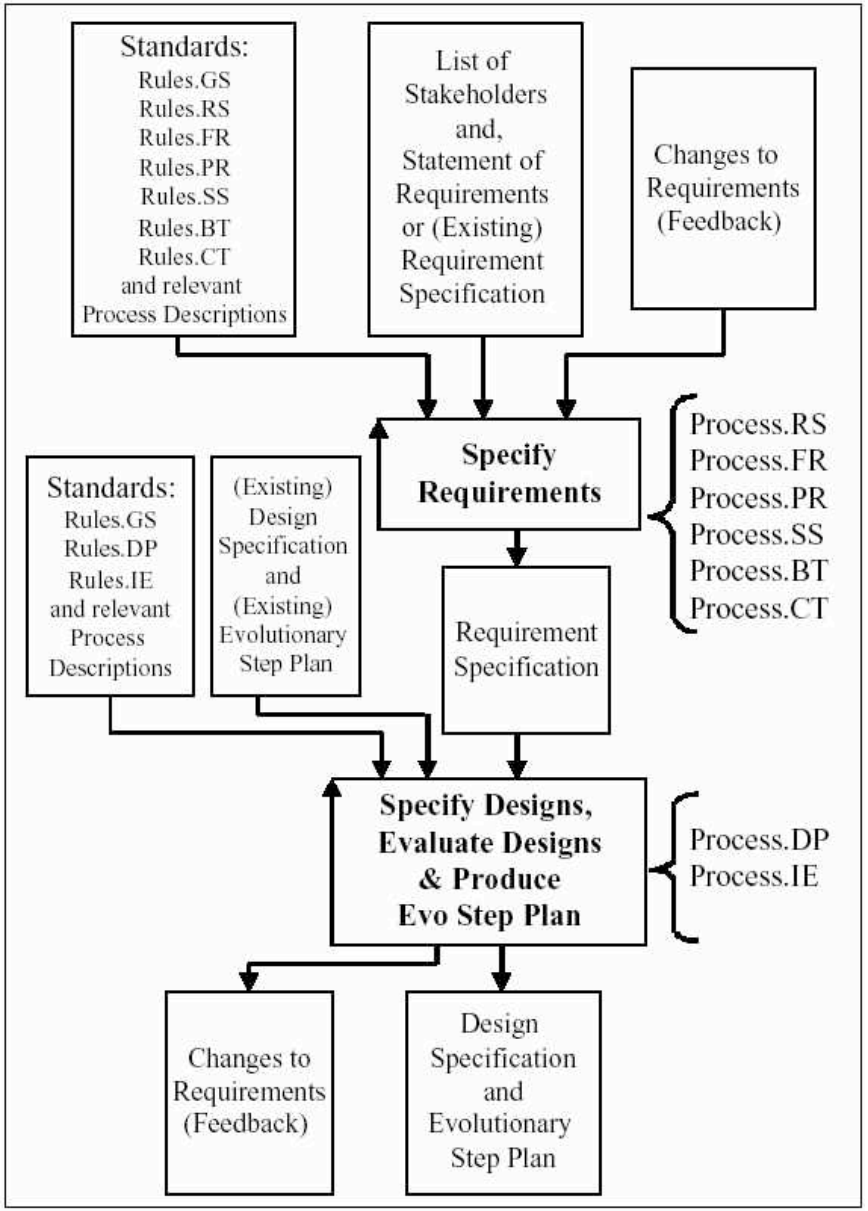


A systems engineering language

(also software, management)

- **Generic Ends-Means process**
- **Well-defined standards**
  - **Specification rules**
  - **Requirements and design processes**
  - **One page - modules**
  - **Reuse of generic standards**
- **Suitable for**
  - **Top management strategy**
  - **Marketing product plans**
  - **Software engineering**
  - **Systems engineering**
  - **Specific engineering**
    - **Aircraft for example**





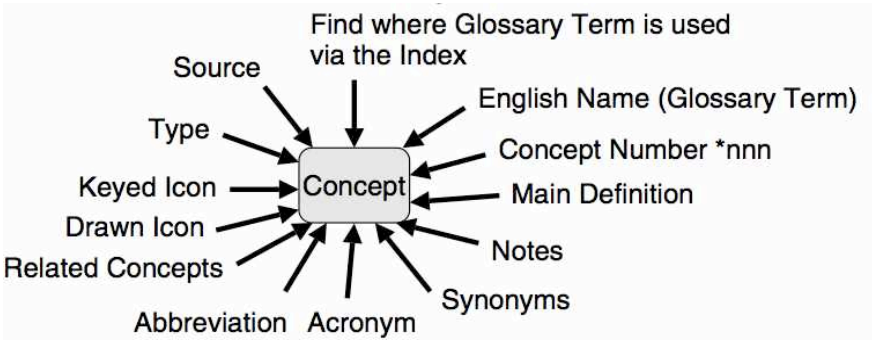
Planguage standards!

Why is Planguage ‘Competitive’?

- It focuses on *high level and critical* **stakeholder needs**
- It is very specific about *when* **results** must be delivered
- It is **quantitative** about all critical values and qualities
- It gives us tools to **prioritize** essentials more intelligently
- It integrates **risk analysis** into all plans dynamically
- It looks at ‘**value for resources’** continuously
- It exploits **realistic** project **feedback** continuously

•! Glossary Purpose.

- •! The central purpose of this Planguage glossary is
  - –! to define **'Concepts' –**
  - –! **not words.**
- •! These concepts have **many 'names'**
  - –! (or 'tags' in Planguage) and attributes.



Requirement
Concept *026 January 23rd 2008

A 'requirement' is a **stakeholder-prioritized future state.**

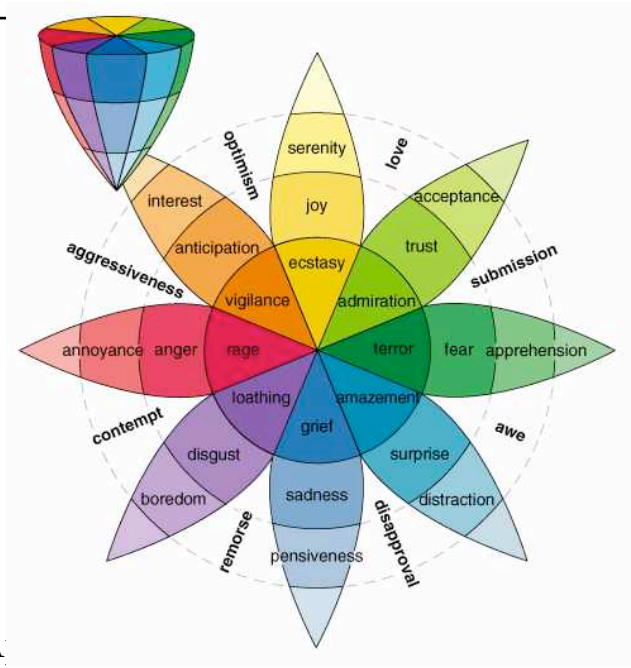
Graphical Language: Notation for Systems Engineering (well maybe 'International' competitiveness)

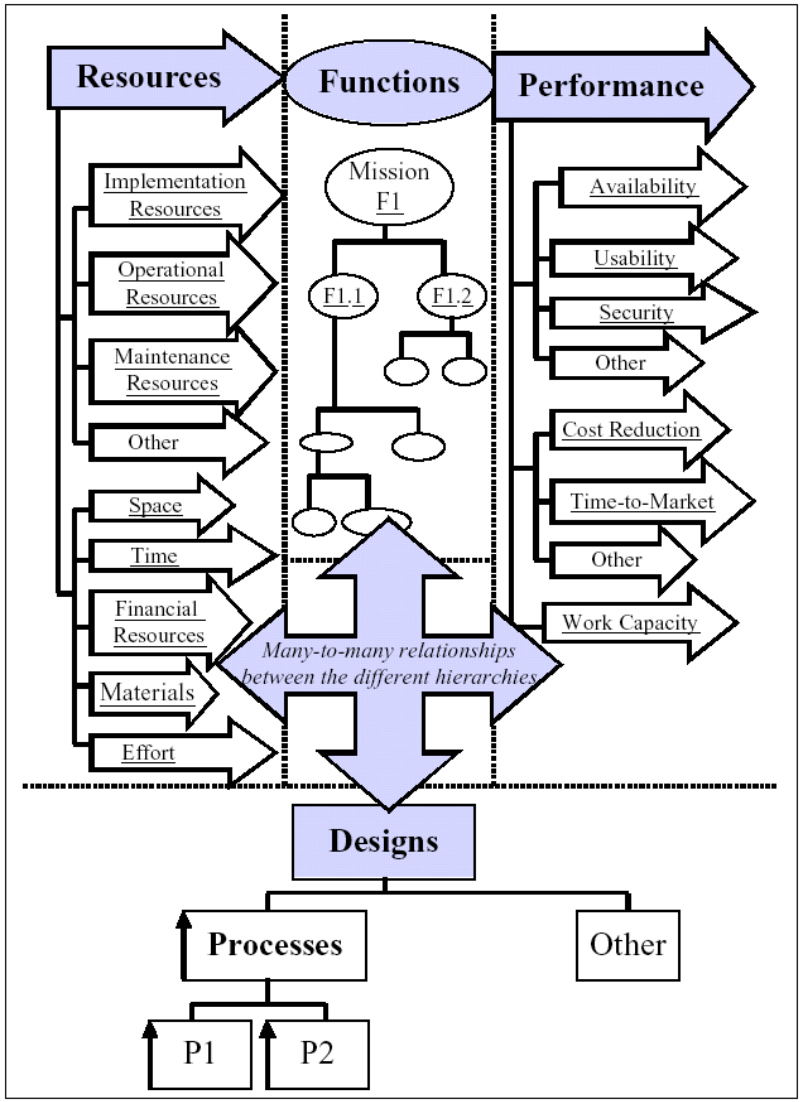
- For many concepts we have defined graphical symbols
- Keyed Icons: <-
  - So that symbols can be keyed in combination with text specification
  - Similar to corresponding drawn icons
- Drawn icons: ←
  - Suitable for graphical presentation
- Why?
  - International language
  - Avoids debates over word choice
  - Short notation

PLANGUAGE TERM	Keyed ICON
Planguage Concept	
Gist:	∑
Ambition Level:	@.∑
Scale:	-|-|-
Meter:	-|?|-
TARGETS	
Goal:	>
Stretch:	>+
Wish:	>?
CONSTRAINTS	
Fail:	>>
Survival Limit:	[ ]
SYSTEM SPACE CONDITIONS	
Time, Place & Event	[qualifier conditions]
Background Information:	
Source:	<-
Comment:	"*text*."
BENCHMARKS	
Past:	<
Record:	<<
Trend:	?<

Control of Multiple dimensions: Performance, Costs, Constraints

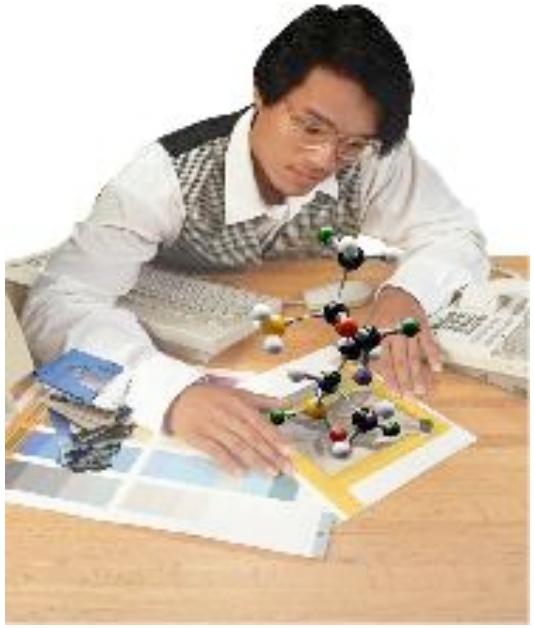
- Planguage specializes in
  - trying to get control over
    - multiple and
    - dynamically changing
    - critical system attributes,
  - through quantified
    - requirement specification,
    - design impact analysis and
    - measurement tactics.
- This helps you compete in a complex environment!

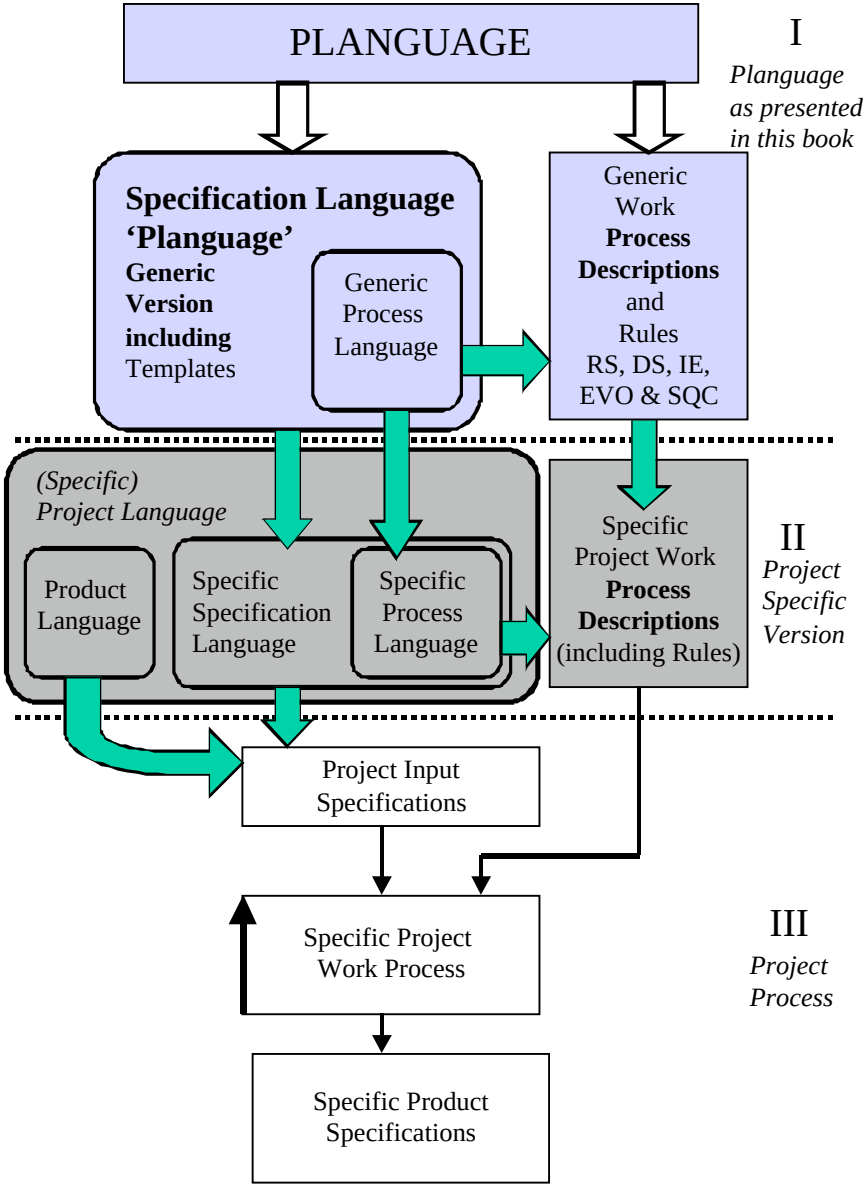




Extendible, Tailorable, Open: Competitive Thru Tailoring

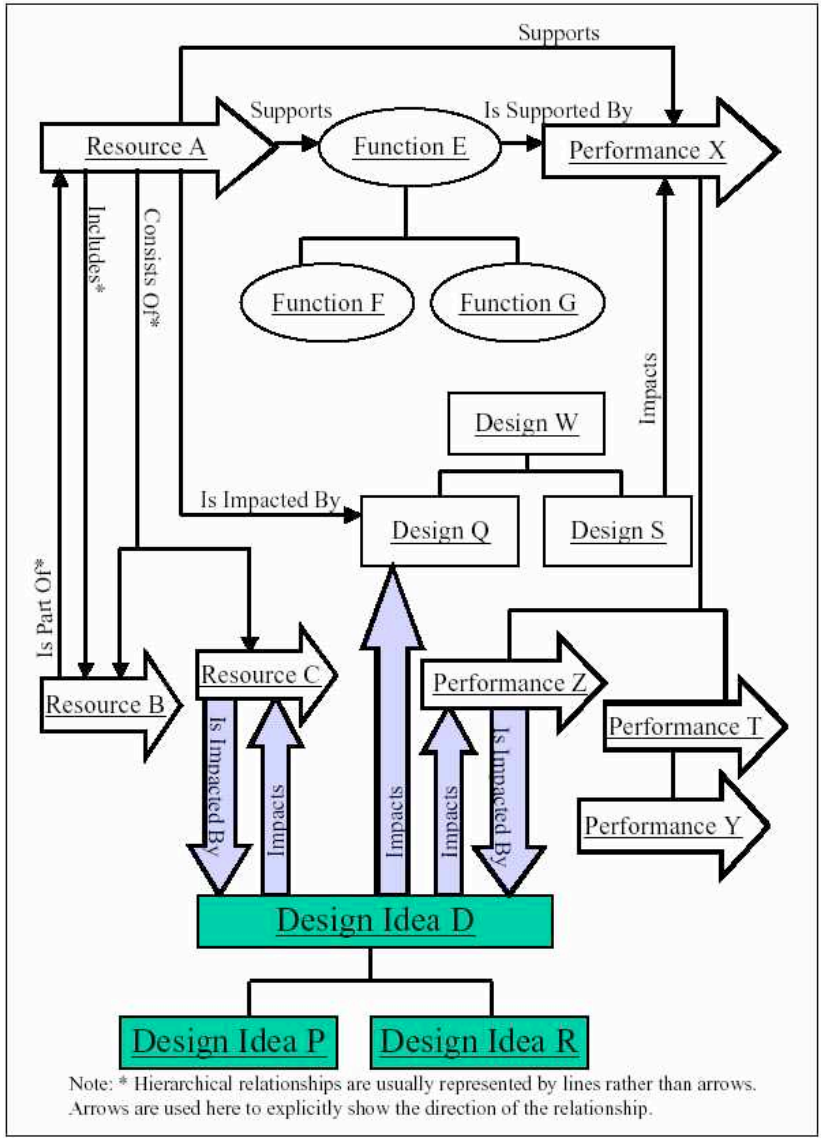
- Planguage:
  - *Free* of cost, & royalties
  - Easy to *extend*
  - Easy to modify *locally*
    - Corporate
    - Project level
    - National language
  - Designed for *re-use* and *tailoring* of reused elements





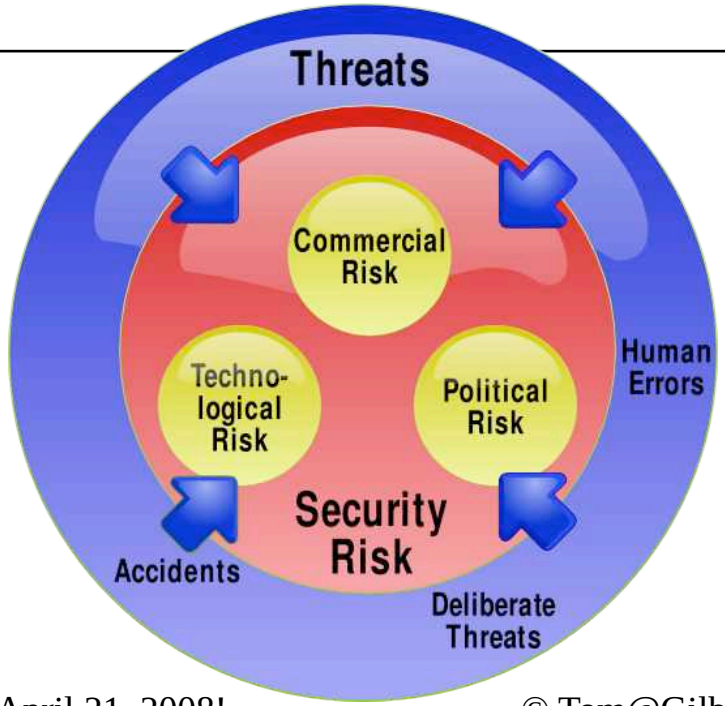
Rich views, traceability, configuration management: Competitive **Insights**

- Some Planguage parameters which define ***relationships***.
  - Authority
  - Source
  - Owner
  - Author
  - Implementer
  - Impacts
  - Supports
  - Supported By
  - Version
  - Derived From
  - Sub-component of
  - Sub-components {list}
  - Dependencies
  - Contract
  - Test Case
  - Scenario
  - Model
  - And more!



Risk Management: Competitive Necessity

- Planguage integrates specific tools for risk specification
  - with more general tools for risk *recognition* and risk *analysis*
  - in a *single integrated* specification language.
- This is a competitive approach to risk management



TEMPLATE FOR FUNCTION SPECIFICATION <with hints>

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

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

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

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

Gist: <Give a 5 to 20 word summary of the nature of this function>.
Description: <Give a detailed, unambiguous description of the function, or a tag reference to someplace where it is detailed. Remember to include definitions of any local terms>.

========== **Relationships** ==========
Supra-functions: <List tag of function/mission, which this function is a part of. A hierarchy of tags, such as A.B.C, is even more illuminating. Note: an alternative way of expressing supra-function is to use Is Part Of>.
Sub-functions: <List the tags of any immediate sub-functions (that is, the next level down), of this function. Note: alternative ways of expressing sub-functions are Includes and Consists Of>.
Is Impacted By: <List the tags of any design ideas or Evo steps delivering, or capable of delivering, this function. The actual function is NOT modified by the design idea, but its presence in the system is, or can be, altered in some way. This is an Impact Estimation table relationship>.
Linked To: <List names or tags of any other system specifications, which this one is related to intimately, in addition to the above specified hierarchical function relations and IE-related links. Note: an alternative way is to express such a relationship is to use Supports or Is Supported By, as appropriate>.

========== **Measurement** ==========
Test: <Refer to tags of any test plan or/and test cases, which deal with this function>

========== **Priority and Risk Management** ==========
Rationale: < Justify the existence of this function. Why is this function necessary? >.
Assumptions: <Specify, or refer to tags of any assumptions in connection with this function, which could cause problems if they were not true, or later became invalid>.
Dependencies: <Using text or tags, name anything, which is dependent on this function in any significant way, or which this function itself, is dependent on in any significant way>.
Risks: <List or refer to tags of anything, which could cause malfunction, delay, or negative impacts on plans, requirements and expected results>.
Priority: <Name, using tags, any system elements, which this function can clearly be done *after* or must clearly be done *before*. Give any relevant reasons>.

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

Competitiveness: Defining it. 1 of 3

Competitiveness

Ambition: Largest 3rd party developer mobile community, demonstrably superior on all **Key Use Cases** to any competitor. <-CEO 19 April 2004.

Enterprise Credentials <-6.8 SPS, Initially. Now defined

Type: Strategic Business Objective.

Version: 4/22/04 9:43 a

Confidentiality: EXAMP

Spec Owner: Simon X

Result Responsible: M

Source: <?>

Past [H1 2004]: ~ 0% <-CEO.

Rationale: there are few enterprises that today use their phones beyond simple voice. <-CEO

Ambition: ensure that Corporate licensees have more than X% of Enterprise deployment,

Scale: % Market Share of defined Enterprise (default All Enterprise) deployment that Corporate Licensees have.

Enterprise: defined as: phones used by Fortune 1000 and SME (Small Medium Enterprise)/SOHO (Small Office Home Office) for services and communication beyond simple voice.

Measurement Process [Longer Term]: <Gartner/IDC/other analyst to produce the stats>.

Measurement Process [H2 short term]: <count the number of network operators actually currently supporting Corporate Licensees in Corporate (hopefully Enterprise) Sales.> In addition, we can look at licensee spend on SXXB (Corporate Enterprise Advisory Board).

Competitiveness: Goals 2 of 3

Goal [H1 2005, Enterprise, If this market actually emerges] 25% ±10%? <-CEO. !

Assertion: this market will suddenly emerge <-CEO!

Goal [H2 2006]: 40%±10%? <-CEO!

Goal [2010] 70% ±20%? <-Guess CEO!

Fail [H2 2006, If this market emerges]: < 25% <-CEO!

Rationale: **(**Fundamental Objective, Big Bill Sidelined**)** ensuring that Big Bill does not secure Dominance (<more than 2x relative market share> <-CEO) in enterprise terminals.!

Value: <Big Bill are not able to leverage their dominance in the corporate sector to break into Enterprise consumer market.> Corporate protects its market share in consumer area.>. <A very big number £> <-CEO!

Competitiveness: Risk 3 of 3

Risks (of not meeting Goal):

R1: pressure to include consumer market PREQs in the product drives out the PREQs required for Enterprise. <-CEO

R2: core enterprise partners fail to invest alongside Corporate. <-CEO

R3: Corporate licensees fail to invest <sufficiently> to support Corporate and the licensees ambitions. Note their marketing people have same conflict as in R1.<-CEO

R4. Corporate geographic footprint blinds it to the Enterprise market. The fact we are strong in Europe, will be in Japan, but small position in USA. <-CEO

R5. Big Owner developments of Enterprise enabling technology are located within Big Owner layers of technology, and are therefore blocked to other Corporate licensees who are not Big Owner licensees.<-CEO

R6. RXX BB are refused to support Corporate OS – Corporate licensees are refused to license RIXX technology because of patent risks. <-CEO

R7: if Big Bill bundling of phones plus Exchange server 2003 is a market-winning proposition. Their classic bundling strategy is applied. <- CEO

R8: others…. Can be added , but not now.

Issues (to be resolved):

I1: can we get Gartner to measure this market in a way we find acceptable (not the PC market tradition they have)? <-CEO

I2: will licensees support SEAB? <-CEO

I3: How will EU anti trust ruling on Big Bill be implemented.? If bundling is blocked, or API's are opened by EU, or then MS proposition is weakened.<- CEO

I4: can Corporate ensure effective cooperation between Series 60 and UIQ to w Enterprise vendors access to the entire orate base with minimum effort? <-CEO tc.

Dependencies (must be in place before we can reach Goal):

D1: none?

Impacted by:

Middleware Provider Support, Operator Endorsement, Analyst Support, SEAB and SEAC Support. <- 2.5 and 2.6 EGMP,

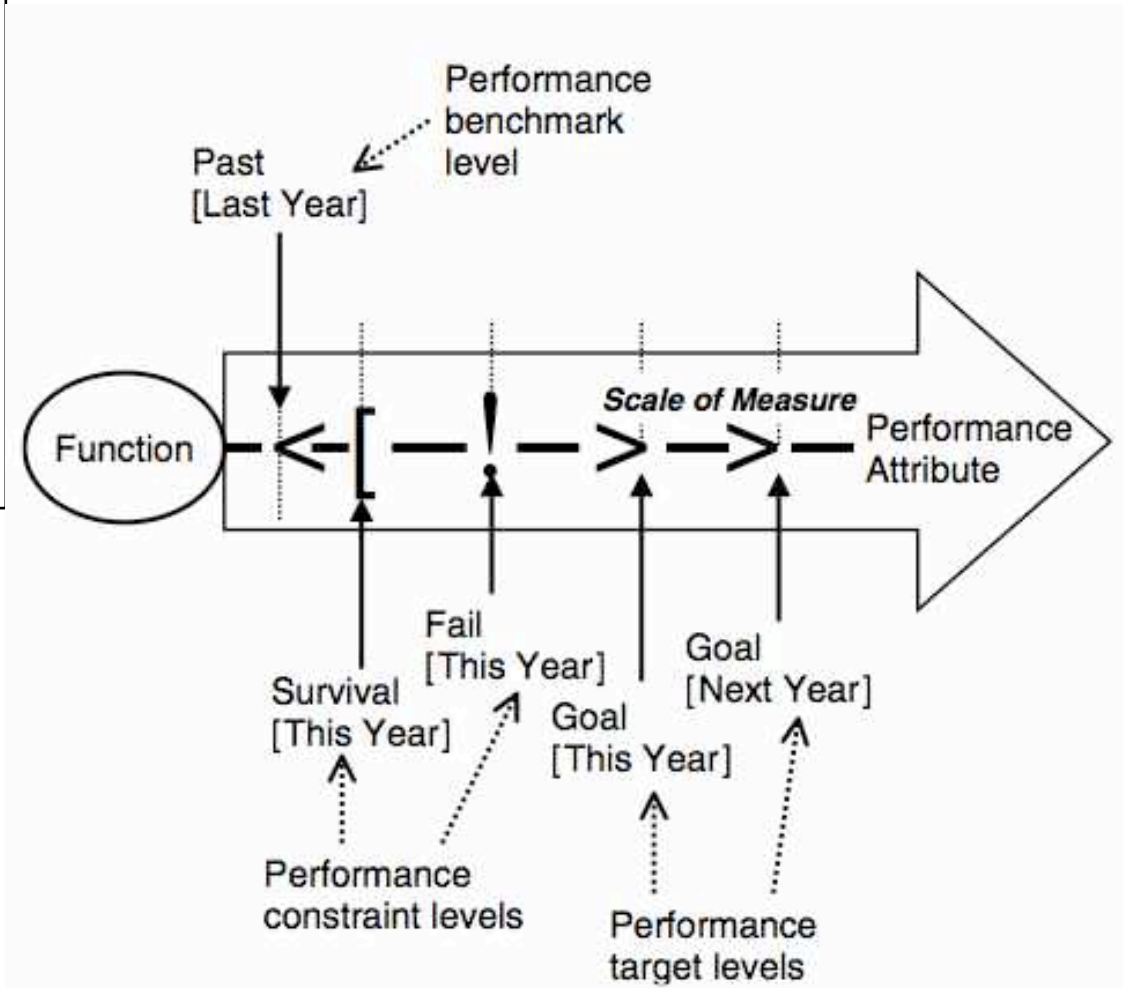
Data Services? <- 2.6 EGMP,

Supports: Big Bill Sidelined

Is Part of Competitiveness

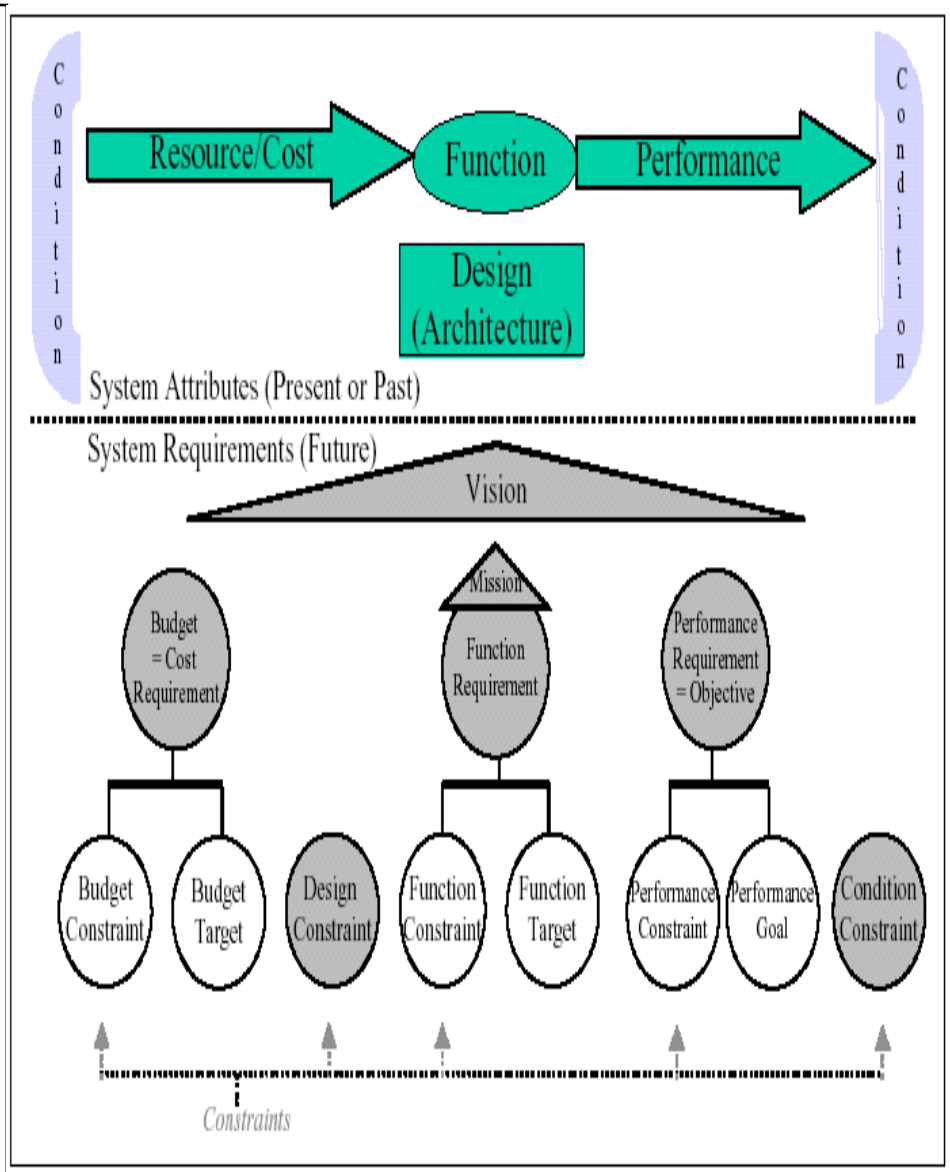
Dynamic Priority Management: Competitive Use of Scarce Resources

- Priority is
  - Claim on scarce or limited resources
- Is a function of
  - Constraint type (Survival, ..)
  - Target type (Goal, ..)
  - Remaining gap to constraint or target level & [qualifiers]
  - Remaining budgeted resources; and their constraint and target levels
- Priority is dynamically computable!
- Priority is also related to other specification parameters such as
  - Authority
  - Sponsor
  - Source



Part 2: Integrating competitive benchmarks and competitive requirement targets

- Systems analysis **benchmarks** are **integrated** with setting future requirements.
- This improves **Competitive Analysis** and Competitive Engineering Specification
  - **Scales**: powerful flexible measures to compete with
  - **Meters**: practical ways to measure performance levels
  - **Benchmarks**: Past, Record, Trend
  - **Targets**: Goal, Stretch, Wish, Ideal
  - **Constraints**: Fail, Survival



‘Function: ‘what a system *does*’.
Requiring ‘Functions’ that are ‘designs’ is uncompetitive

Function
Symbol =!
‘Oval!

Function !

Design
Symbol =
Rectangle

- Functions are *often confused* with *other* specifications, like:
  - **Features**’ (*innovations*, compared with other systems)
  - ***Means* to ends** (like ‘designs’, ‘architecture’, ‘strategies’)
  - **Use *Cases*** (human to system interaction sequences,
    - which may be partly ‘analysis’ (‘what is’),
    - or ‘design (what we *might* want).
  - **DANGER**: If you accept, or cause, the *confusion*,
    (requiring designs, that are not *really* ‘required’)
    - You are likely to get *uncompetitive* designs,
      - Meaning you get worse performance and costs,
        - Than you *could* have gotten.
      - Planguage is extremely conscious of the difference,
        - and tries to make sure *you do get your competitive opportunities*.

Function !

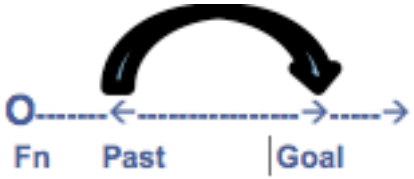
?

Design

‘Requirement’: Defined

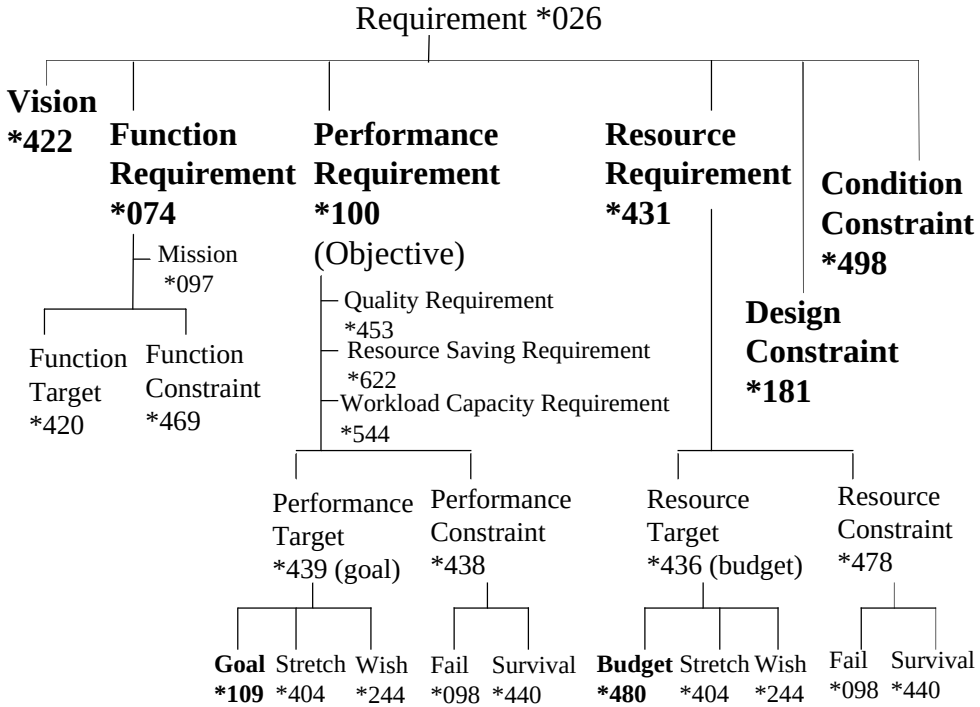
Concept *026 Version January 23rd 2008

- A ‘requirement’ is a
  - **“stakeholder-prioritized future state”.**



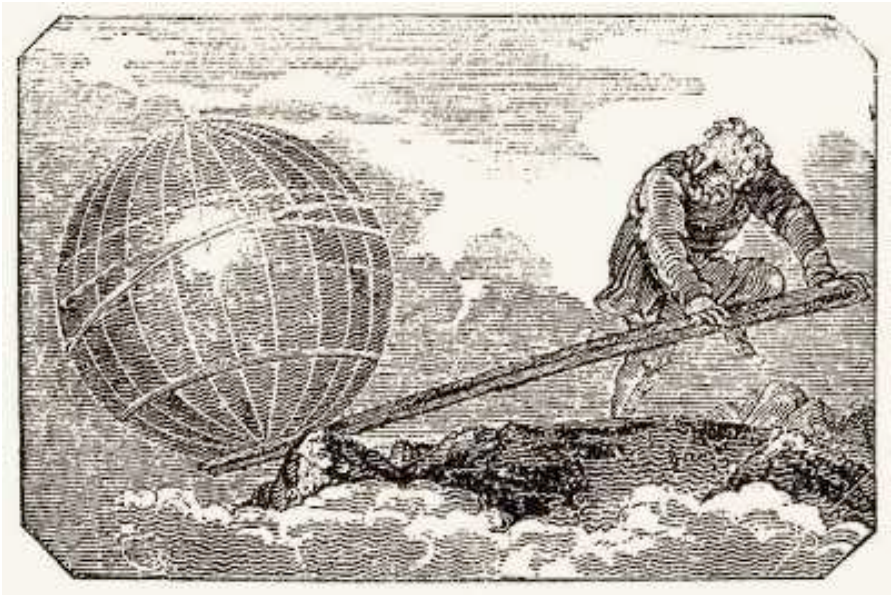
- **Some consequences of this definition:**
  - **requirements are *not* ‘absolute’**
  - **a requirement’s effective priority’ is *variable*, and depends on *many* factors, like**
    - **Value of doing it, cost of doing it, related constraints,**
    - **stakeholder power, formal requirement inclusion.**
  - **Planguage helps you intelligently manage requirement priorities, so that you get maximum value for your limited resources (= ‘competitiveness’).**

Some!
Formally !
Defined!
Requirement!
Concepts and!
types!

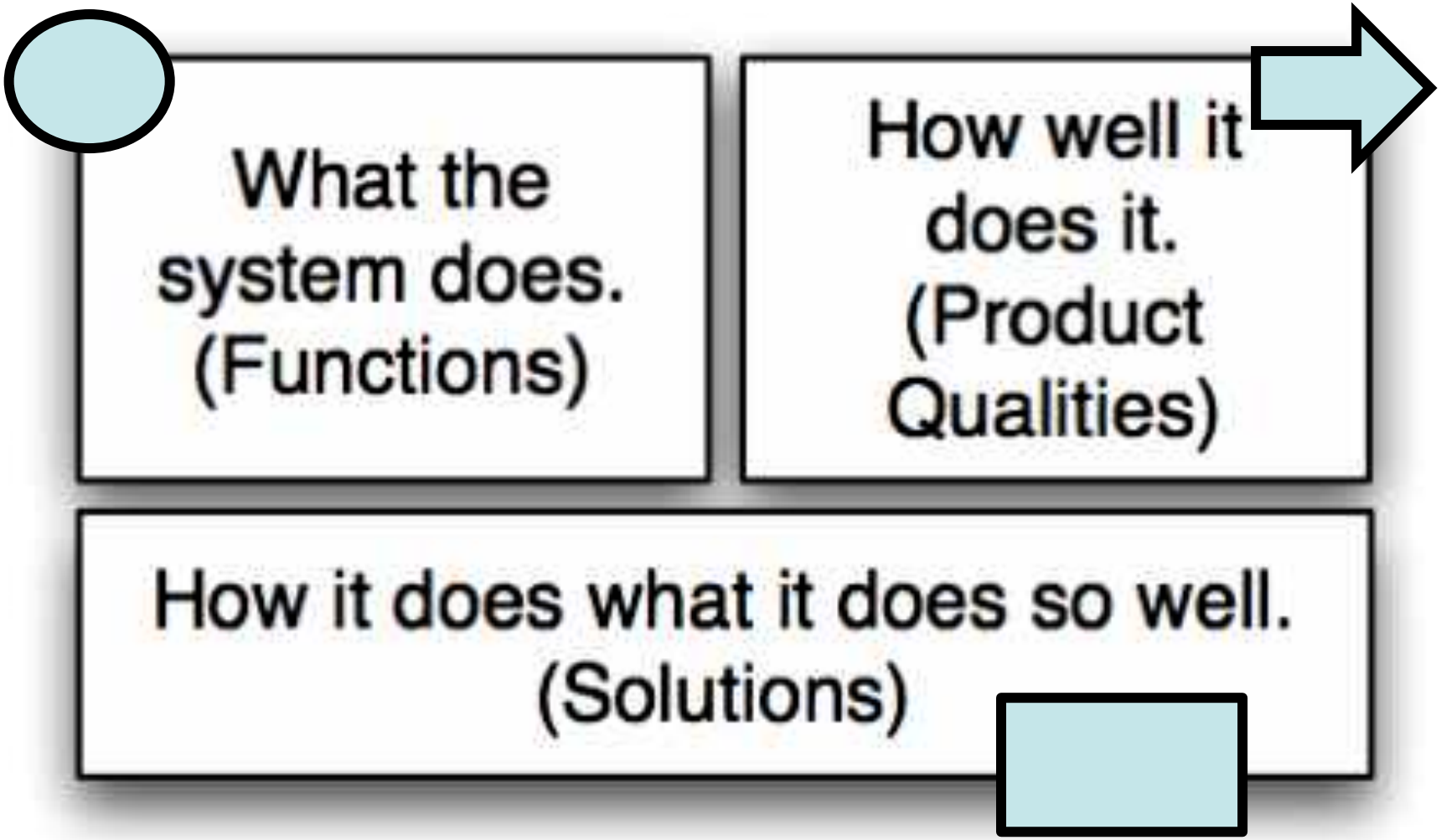


‘Requirement’ – a competitive concept

- A clear understanding and agreement about what a ‘requirement’ is
  - Allows you to be more competitive
  - by focusing on
    - REAL COMPETITIVE NEEDS
    - At a competitively high level
      - Where the power and leverage and decision-making is.



3 views of a system: Powerful distinctions



Systems analysis benchmarks are integrated with setting future requirements.

Adaptability:

Type: Quality Requirement.

Scale: the calendar time in hours needed to re-configure the defined [Base Configuration] to any other defined [Target Configuration] using defined [Methods] and defined [Reconfiguration Staff].

Expert Reconfiguration: Defined As:

{Base Configuration = Novice Setup,
Target Configuration = Expert Setup,
Methods = Selection of Library Reconfiguration Process,
Reconfiguration Staff = Qualified Expert}.

======== Benchmarks ==========

Past [Expert Reconfiguration, Version 0.3, Asian Market]: < 1 hour.

========= Goals (Performance Targets)==========

Authority [Goals]:Federal Drug Administration.

Goal [Expert Reconfiguration, Deadline = Version 1.0]: < 0.5 hours.

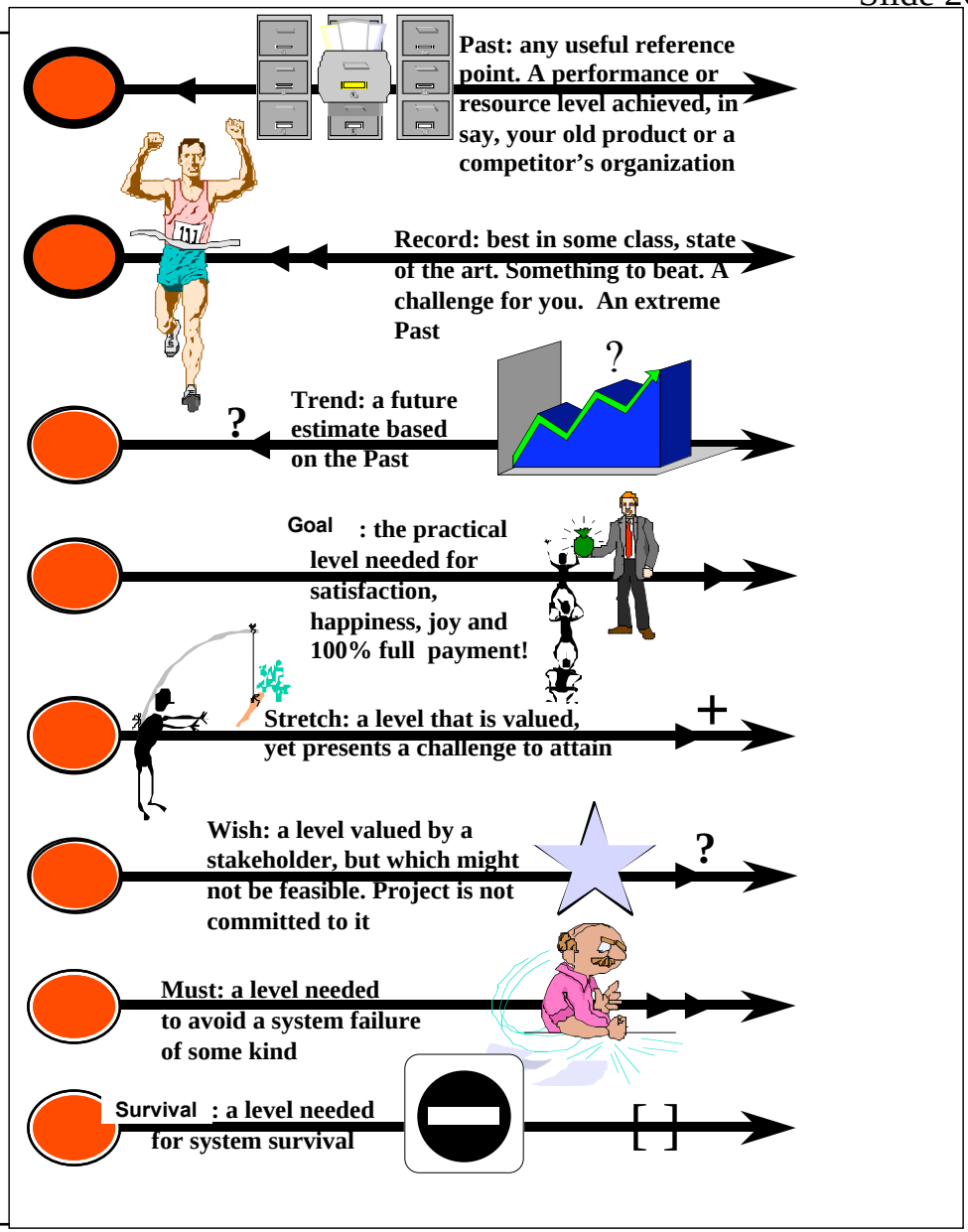
Goal [Expert Reconfiguration, Deadline = Version 2.0]: < 0.1 hours.

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

Fail [All USA Products]: < 0.7 hours.

Fail [Expert Reconfiguration, Deadline = Version 2.0]: < 0.5 hours.

Survival [Expert Reconfiguration, European Market]: < 1 working day.



Benchmark/Requirement Integration
improves Competitive Analysis and Competitive Engineering Specification

- Competitive Analysis
  - Make sure your own and competitor levels **(Past, Record)** are
    - analyzed and specified
    - together with future requirements (**Trend**)
- Competitive Engineering
  - Make sure you not only specify the balanced '**Goal**'
  - but that marketing information about '**Wish**' is captured.
    - Even if they cannot be satisfied just now!
  - Make sure that the engineer is challenged by a '**Stretch**' goal

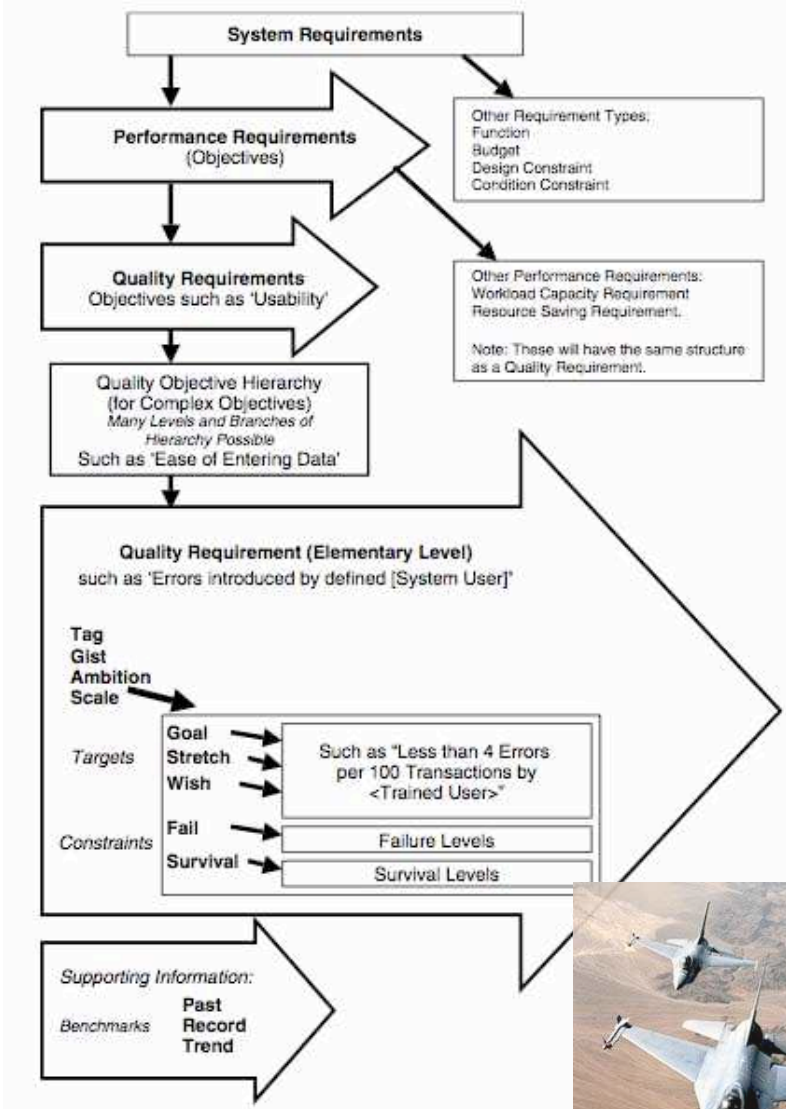


Figure 4.7
Requirement specification hierarchy for a quality requirement.

Scale: The Quantification Foundation

Scale!

-|-|-! !

Concept *132 August 17, 2004!

- "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.

User Friendly:

Type: Quality Requirement. **“Teotihuacan”**

Ambition: To consistently exceed Competitor’s ease of learning.

Scale: Time to Master
a defined [Task]
by defined [Learner].

Meter: <Use good academic practice, do at least 10 Tasks, with at least 5 Learner Types and at least 50 people>.

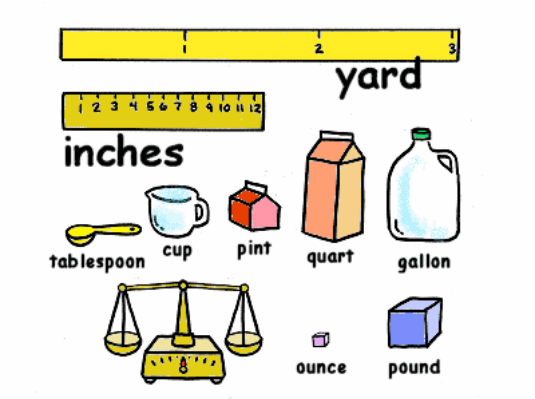
Record [Competitor AA, Product XYZ, Task = Dial Out, Learner = Novice]: 2 minutes <- Our current tests.

Goal [Our Company, Product ABC, Task = Dial Out, Learner = Novice]: < 10 seconds <- Marketing Requirement 4.5.7.

Master: Defined as: ability to pass a suitable approved test.

Defined ‘Scale’ : Basic Competitive Tool

- Demands comparative **thinking**.
- Unambiguously **clear**
- Team **Aligned** with Business



Real Example of ***Lack*** of Scales

*1. Central to The Corporations business strategy is to be the world’s **premier** integrated_<domain> service **provider**.*

*2. Will provide a much more efficient **user** experience*

*3. Dramatically scale back the **time** frequently needed after the last data is acquired to time align, depth correct, splice, merge, recompute and/or do whatever else is needed to **generate** the desired **products***

*4. Make the system much **easier** to **understand** and **use** than has been the case for previous system.*

*5. A primary goal is to provide a much more **productive** system **development** environment than was previously the case.*

*6. Will provide a richer set of functionality for **supporting** next-generation logging **tools** and applications.*

*7. **Robustness** is an essential system requirement (see rewrite in example below)*

*8. Major improvements in **data quality** over current practices*

This lack of clarity cost them $100,000, 000!

Meters: practical ways to measure performance levels.
To Give us facts and know how to compete better

Meter **-|?|-** ***Concept *093 April 18, 2003!***

- 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, 1986 (Out of the Crisis, MIT Press)

Repair:

Ambition: Improve the speed of repair of faults substantially, under given conditions.

Scale: Hours to repair or replace, from fault occurrence to when customer can use faultlessly, where they intended.

Meter [Product Acceptance]: A formal test in field with at least 20 representative cases,

[Field Audit]: Unannounced field testing at random.

========== Benchmarks ==========

Past [Product = Phone XYZ, Home Market, Qualified Dealer Shop]:
{0.1 hours at Qualified Dealer Shop +
0.9 hours for the Customer to transit to/from Qualified Dealer Shop}

Record [Competitor Product XX]: 0.5 hours average.
"Because they drive a spare to the customer office."
Trend [USA Market, Large Corporate Users]: 0.3 hours. "As on-site spares for large customers."

========== Targets ==========

Goal [Next New Product Release, Urban Areas, Personal Users]: 0.8 hours in total,
[Next New Product Release, USA Market, Large Corporate Users]: 0.2 hours
<-Marketing Requirement, 3 February This Year.

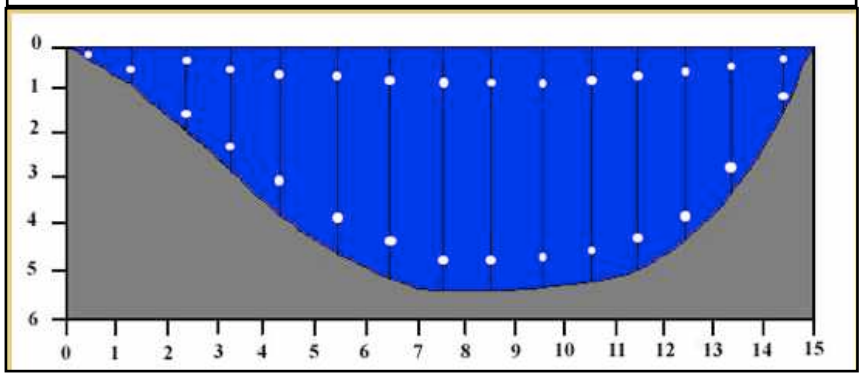
========== Constraints ==========

Fail [Next New Product Release, Large Corporate Users]: 0.5 hours or less on average
<-Marketing Requirement, 3 February This Year.

Meter: The Measuring Process:
Competitive Feedback Early and Frequently

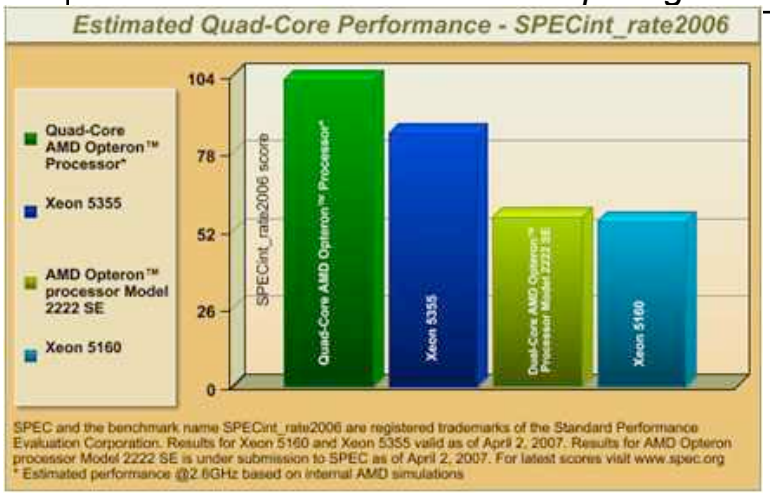
Stream gaging along the Verde River, Arizona

Diagram of a stream cross section showing the location of velocity measurements (white dots) that must be acquired during gaging.



Benchmarks: Past, Record, Trend
Benchmarks tell us where we are, or will be, in relation to competitors

- **Past**: A relevant benchmark level already achieved by an existing system (our own, competitive, or any other system) that is worth consideration.
- ***Record****: A 'Past', which is the best known result [in some defined area]. A 'state-of-the-art' value.*
- ***Trend****: An extrapolation of past data, trends and emerging technology to a defined [time and place].*
  - *Aside from our own project's plans to improve this level, what future levels are likely to be achieved by others?*
  - *What will we be competing with?*



Usability [New Product Line, Major Markets]:

Ambition: To achieve a low average time-to-learn to use our telephone answerer, under various conditions.

Scale: Average number of minutes for defined [representative user and all their household family members over 5 years old] to learn to use defined [basic daily use functions] correctly.

Meter [Product Acceptance]: A formal test in field with at least 20 representative cases,

[Field Audit]: Unannounced field testing at random.

========== Benchmarks ==========

Past [Product XYZ, Home Market, People between 30 and 40 years old, in homes in Urban Areas, <For one explanation & demo>]: 10 minutes.

Record [Competitor Product XX, Field Trials]: < 5 minutes?> <- one single case reported,

Trend [USA Market, S Corporation, By Initial Release]: 10 seconds <- Public Market Intelligence Report.

======== Constraint ==========

Must [Next New Product Release, Children over 10]: 5 minutes <- Marketing Requirements 3 February Last Year.

========== Targets ==========

Plan [Next New Product Release, Urban Areas, Personal Users]: 5 minutes total,

[Next New Product Release, USA Market, Large Corporate Users]: 5 minutes <- Marketing Requirements 3 February Last Year.

Stretch [Next Year]: (Record - 10%).

Benchmarks **are the *basis* for setting future competitive goals**

- Benchmark Levels
  - Are 'systems analysis
  - Determine where **you** are '**now**'
    - **Past, Now**
  - Where you might be in **future**
    - **Trend**
  - Where **competitors** are now
    - **Past, Record**
  - Where they might be in the **future**
    - **Trend**
  - Can tell us '**state of the art**'
    - **Record**

Wooden sankofa bird –!
From the country of Ghana!
a wooden representation of!
the fabled Sankofa Bird.!
The Sankofas' head is!
always turned backwards!
thus "facing the past."!
The Sankofa represents!
the old African adage!
"Always remember the past!
for therein lies the future!
if forgotten..."!
We are destined to repeat it.!

Targets: Goal, Stretch, Wish, Ideal
The Competitive Requirement or Need

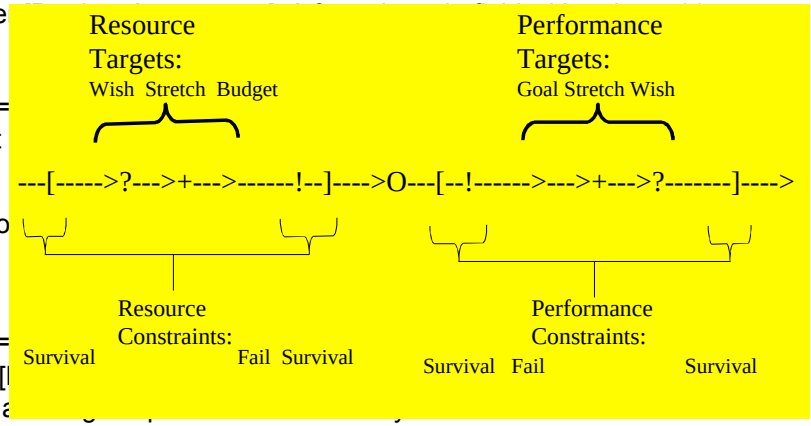
- **Goal:** *A future required level*
  - *under [defined conditions], which*
  - *at **least has to be achieved to claim success** in meeting a requirement.*
  - *A signal to stop investing in levels better than this level;*
    - *because the value gained is insufficient to justify additional costs.*
- ***Budget**: a 'Goal' level for **costs**.*
- ***Stretch**: A future desired and valued level, under [defined conditions], which **is designed to challenge** people to exceed Plan levels.*
- ***Wish**: A future desired level, which **is valued by a stakeholder.***
  - *The requirement is **not planned or promised yet**;*
  - *due to technical or cost reasons – or lack of evaluation,*
  - *but it is recorded, and kept in the requirement database (even if not acceptable now),*
  - *so that it can be borne in mind as a future competitive opportunity.*
- ***Ideal:** a future desired level which is **perfect**.*

Usability [New Product Line, Major Markets]:

Ambition: To achieve a low average time-to-learn to use our telephone answerer, under various conditions.

Scale: Average number of minutes for defined [representative user and all their household family members over 5 years old] to learn to use defined [basic daily use functions] correctly.

Mete



========= Targets ================

Goal [Next New Product Release, Urban Areas, Personal Users]: 5 minutes total,

[Next New Product Release, USA Market, Large Corporate Users]: 5 minutes <- Marketing Requirements 3 February Last Year.

Stretch [Next Year]: (Record - 10%).

Wish [Ultimately] <few seconds>

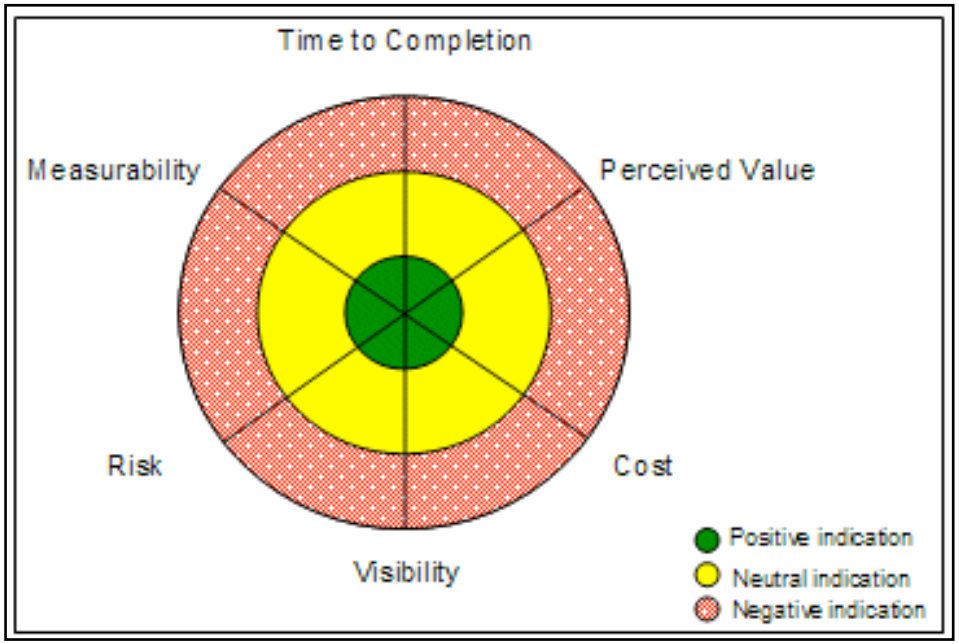
Ideal: 0 seconds.

Targets
Your Vision of Being Competitive

Competitive Levels of performance

- Speculation, Subjective
- Can be *adjusted* as we learn what is competitive
- Have unknown *costs*
- Have unknown *side effects*
- Can be adjusted *as we learn* costs and effects
- Priority of a target *varies* depending on
  - Costs
  - Many factors like power, value, policy

Target Priority Varies



Targets
Numeric Points On A Scale:
Your Vision of Competitiveness

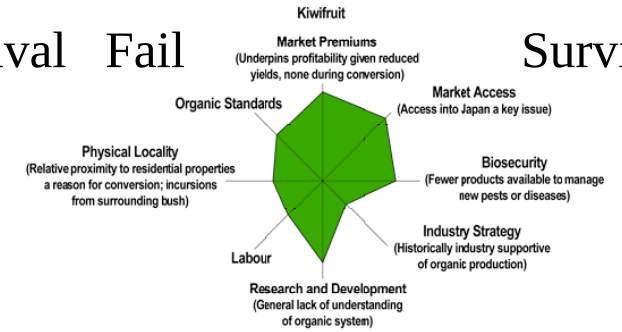
Resource Targets:
Wish Stretch Budget

Performance Targets:
Goal Stretch Wish

---[----->?--->+--->------!--]---->O---[--!------>--->+--->?-------]---->

Resource Constraints:
Survival Fail Survival

Performance Constraints:
Survival Fail Survival



Some Constraints: Respect while being competitive

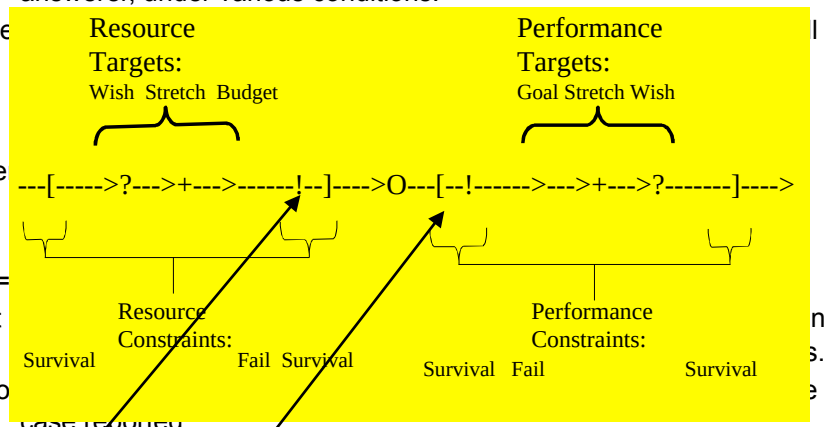
Constraint **Levels**: Fail, Survival: Respect Constraints - to avoid Failure

- **Fail** Concept *098 April 21, 2003
  - '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 level specifies a point at which a system or attribute failure state can occur.
  - A single specified number (like Fail: 90%) is assumed to be the leading edge of a Failure Range.
- **Survival** **Concept *440 March 3, 2003!**
  - 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.

Usability [New Product Line, Major Markets]:

Ambition: To achieve a low average time-to-learn to use our telephone answerer, under various conditions.

Scale



Mete

====

Past

Reco

case reported,

[USA Market, S Corporation]: 10 seconds <- Public Market Intelligence Report.

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

Fail [Next New Product Release, Children over 10]: 5 minutes

<- Marketing Requirements 3 February Last Year.

Survival [Next New Product Release, Children over 10]: 10 minutes

========= Targets ================

Goal [Next New Product Release, Urban Areas, Personal Users]: 5 minutes total,

[Next New Product Release, USA Market, Large Corporate Users]: 5 minutes <- Marketing Requirements 3 February Last Year.

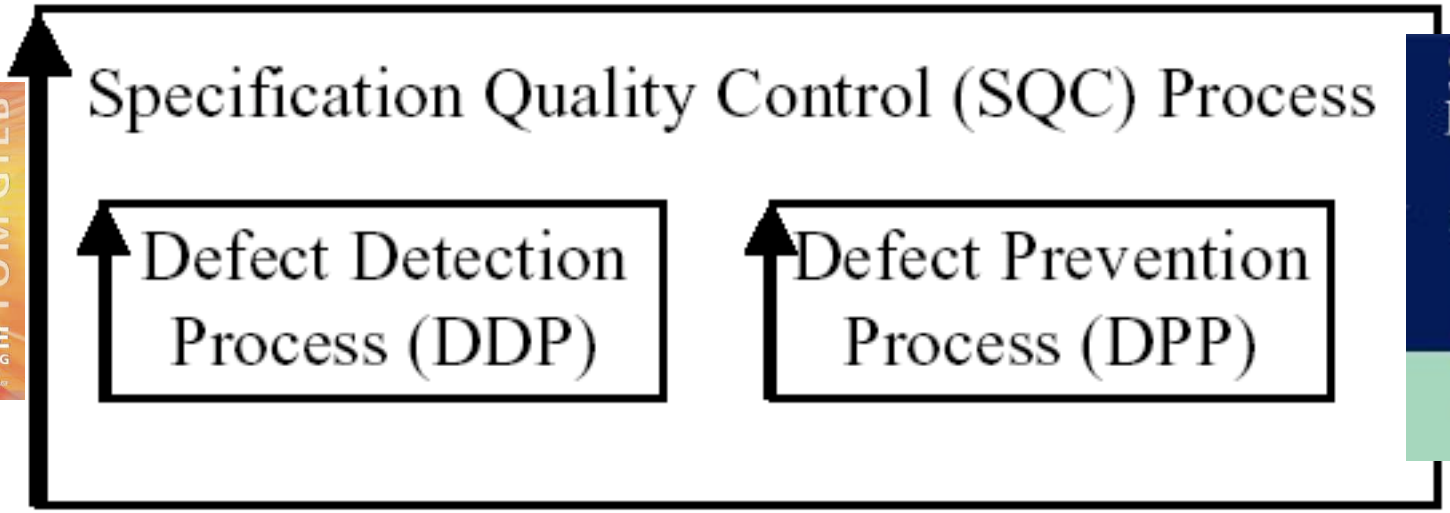
Stretch [Next Year]: (Record - 10%).

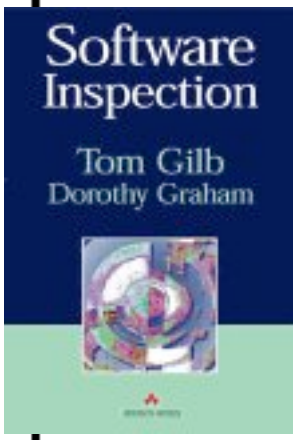
Enthoven on Numbers

- "Numbers are a part of our language.
- Where a quantitative matter is being discussed,
  - the greatest clarity of thought is achieved by using numbers
  - instead of avoiding them,
  - *even when uncertainties are present.*
- *This is not to rule out judgment and insight.*
  - *Rather, it is to say, that*
  - *judgments and insights need,*
  - *like everything else,*
  - *to be expressed with clarity*
  - *if they are to be useful."*
- Alain Enthoven, June 1963, Naval War College, Newport Rhode Island (see note for more detail), Hughes98, Rescuing Prometheus p164

See the note for more detail on Enthoven!

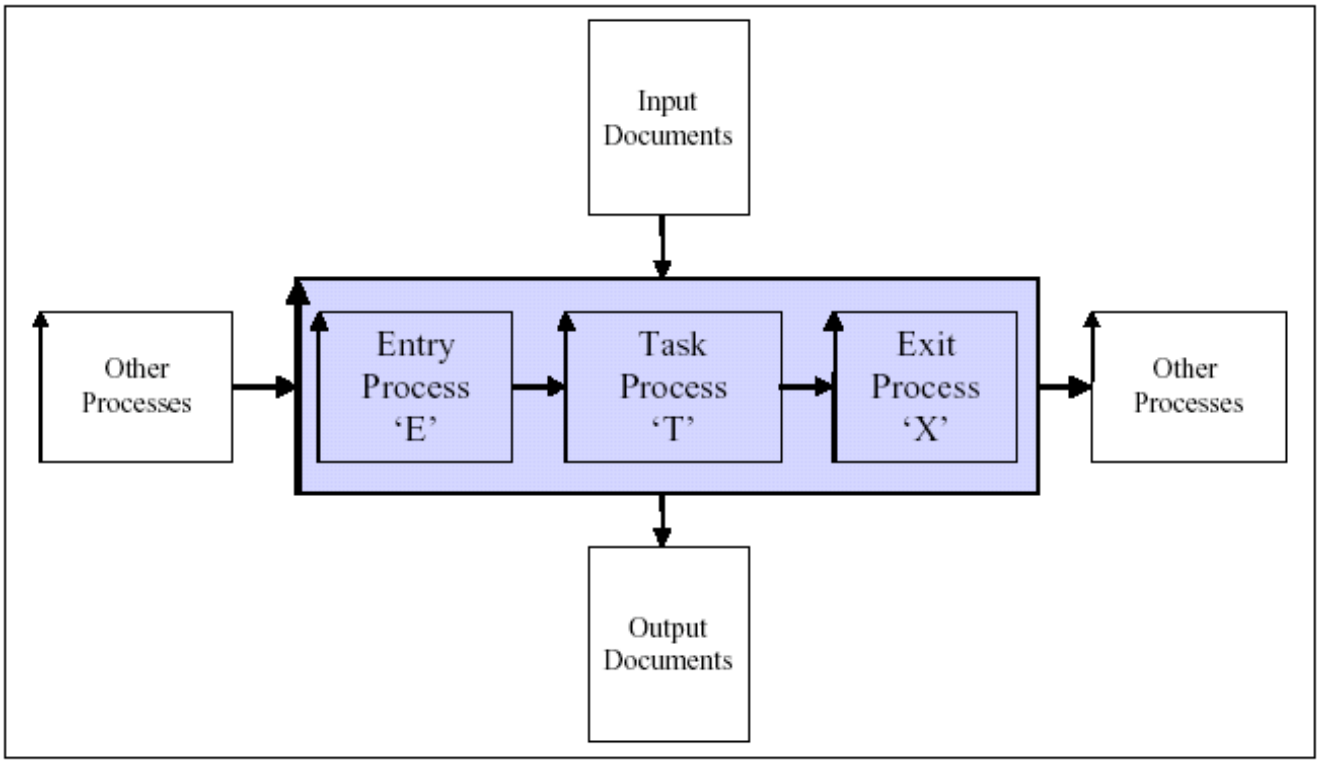
Part 3: Quantified Quality Control of specifications
Competing By Stopping "Garbage In" Earlier





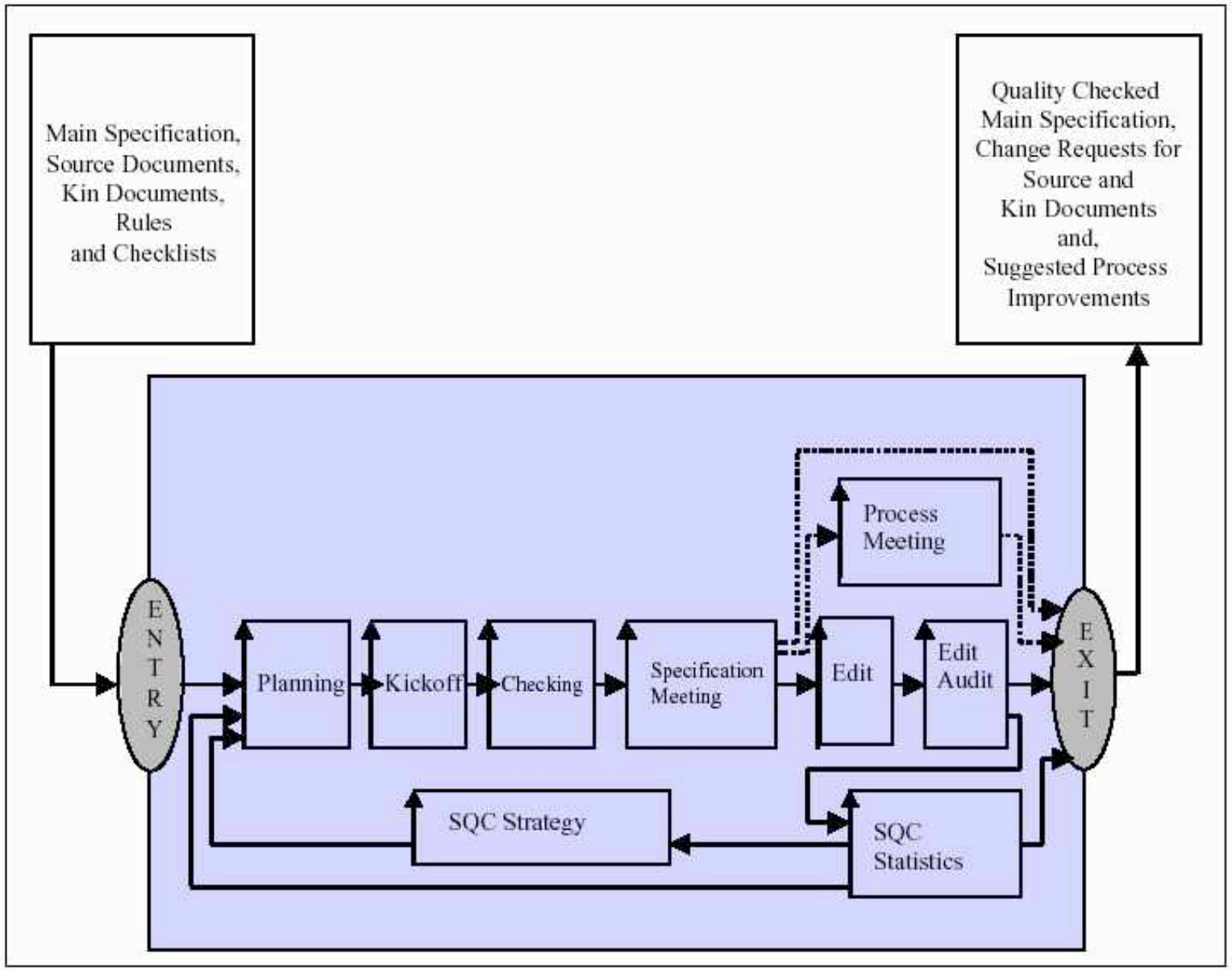
- Quality Control of Specification (SQC)
- The quantified Exit and Entry controls
- Reviewing the Quality of a specification's 'Competitiveness'
- How does Planguage help QC?
- How does Planguage help Reviews?
- How does QC impact competitiveness?

Quality Control of Specification (SQC)

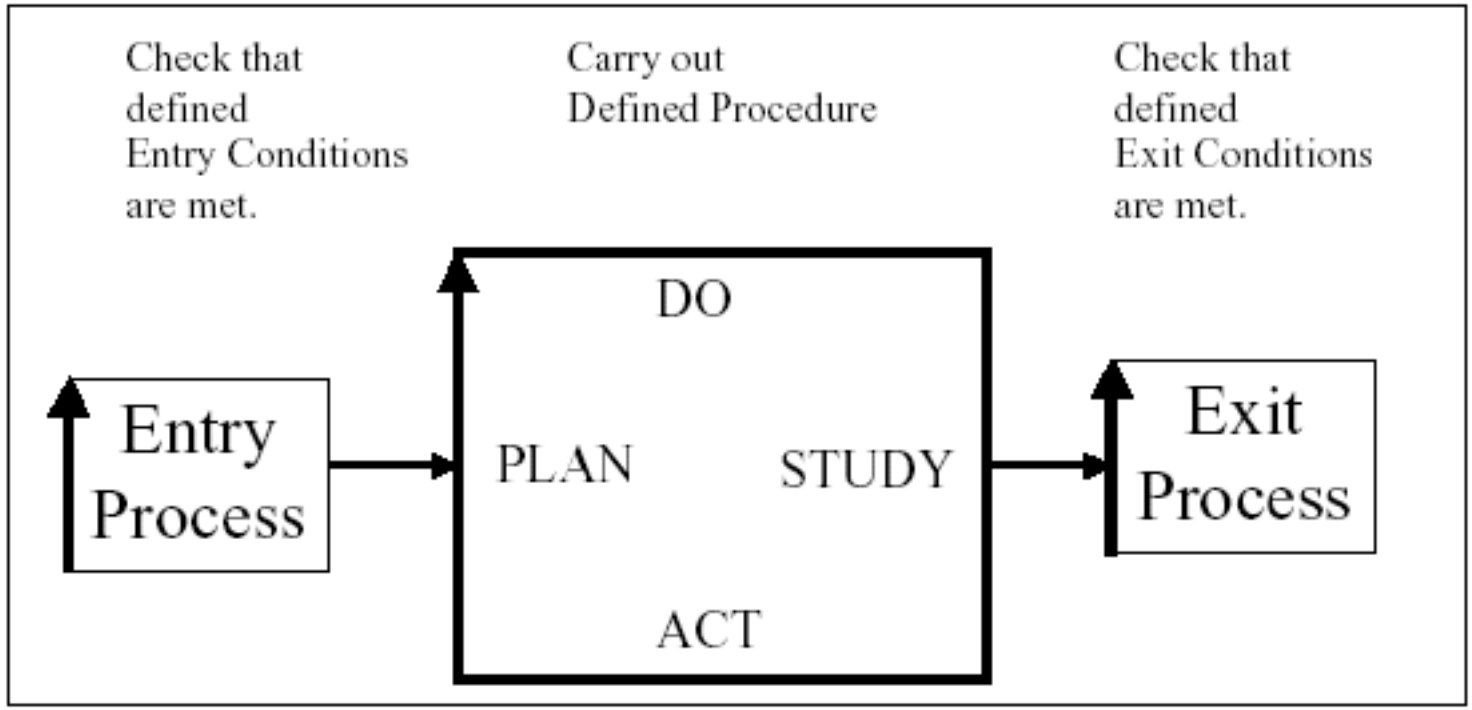


- Spec QC is done
  - when the input (other) work process meets entry conditions (E)
  - According to a defined QC process (T)
  - And is released to other process when exit conditions are met (X)
  - And is done by comparison with other related documents and spec rules (Input)
  - Producing reports and process control statistics (Output)

Quality Control of Specification: Detail (2)

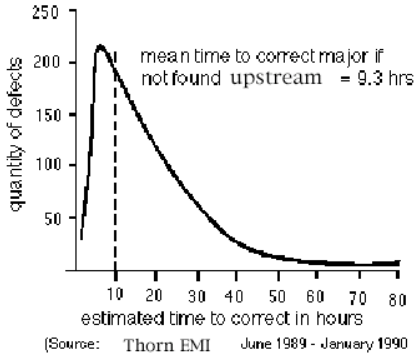


The quantified Exit and Entry controls



- ! Entry and Exit Condition example:
- ! Maximum estimated 1.0 Major defects per logical page remaining.

The quantified Exit and Entry controls (2)



Assumptions:!
1) 30 major defects/page have been found during SQC. !
2) Your SQC effectiveness is 60% and your SQC is a statistically stable process). !
3) One sixth of your attempts to fix defects fail (One sixth is average failure to fix.) !
4) New defects are injected during your attempts to fix defects at 5%. !
5) The uncertainty factor in the estimation of remaining defects is ± 30%. !

Probably remaining major defects in each (logical) page = !
'probably unidentified majors' + 'bad fix majors' + 'majors Injected' !

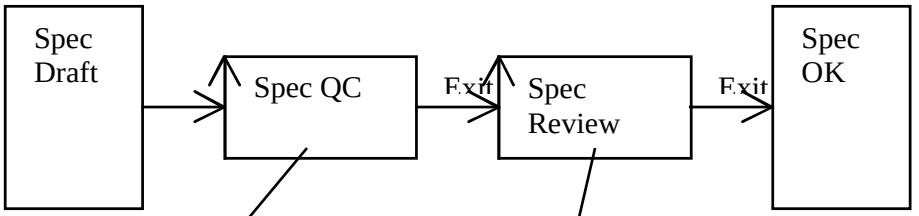
Let E = Effectiveness expressed as a percentage (%) = 60% !
Probably unidentified majors = major defects acknowledged-by-editor for each page at Edit * (100 – E) / E !
= 30 major defects/page found * (100 - 60) / 60 = **20 major defects/page.** !

Bad Fix Majors = One sixth of fixed majors = So, of 30 attempted fixes, !
! **5 major defects in each page** are not fixed. !
Majors Injected = 5% of majors attempted to be fixed = **1.5 major defects/page.** !

Probably remaining major defects/page = 20 + 5 + 1.5 = **26.5 remaining major defects/page** !
Taking into account the uncertainty factor of ± 30% and rounding down to the nearest whole !
number gives **26 ± 7 Remaining Major Defects/Page** !
(Minimum = 19, Maximum = 33 remaining major defects/page). !

Reviewing the Quality of a specification's 'Competitiveness' Slide 45!

- Entry Condition:
  - Low-defect exit from *Specification* Rules QC
    - So it is complete, clear, consistent, correct
- Different people (Senior)
  - Different Rules, ask them
    - About idea value
    - About other investments
    - About competition
    - About economics
    - About risks
- Different Evaluation
  - Not 'defects'
    - (Rules decide!)
  - Go or no-go to next stage of development
    - (Exit, numeric objective)
  - Responsible recommendations
    - What to do if 100 Majors/Page?
  - Status determination
    - (Approved, Clarity Exit, Content Exit, Not Exit, Draft Not Reviewed…)



QC & Spec Rules (Clarity)!

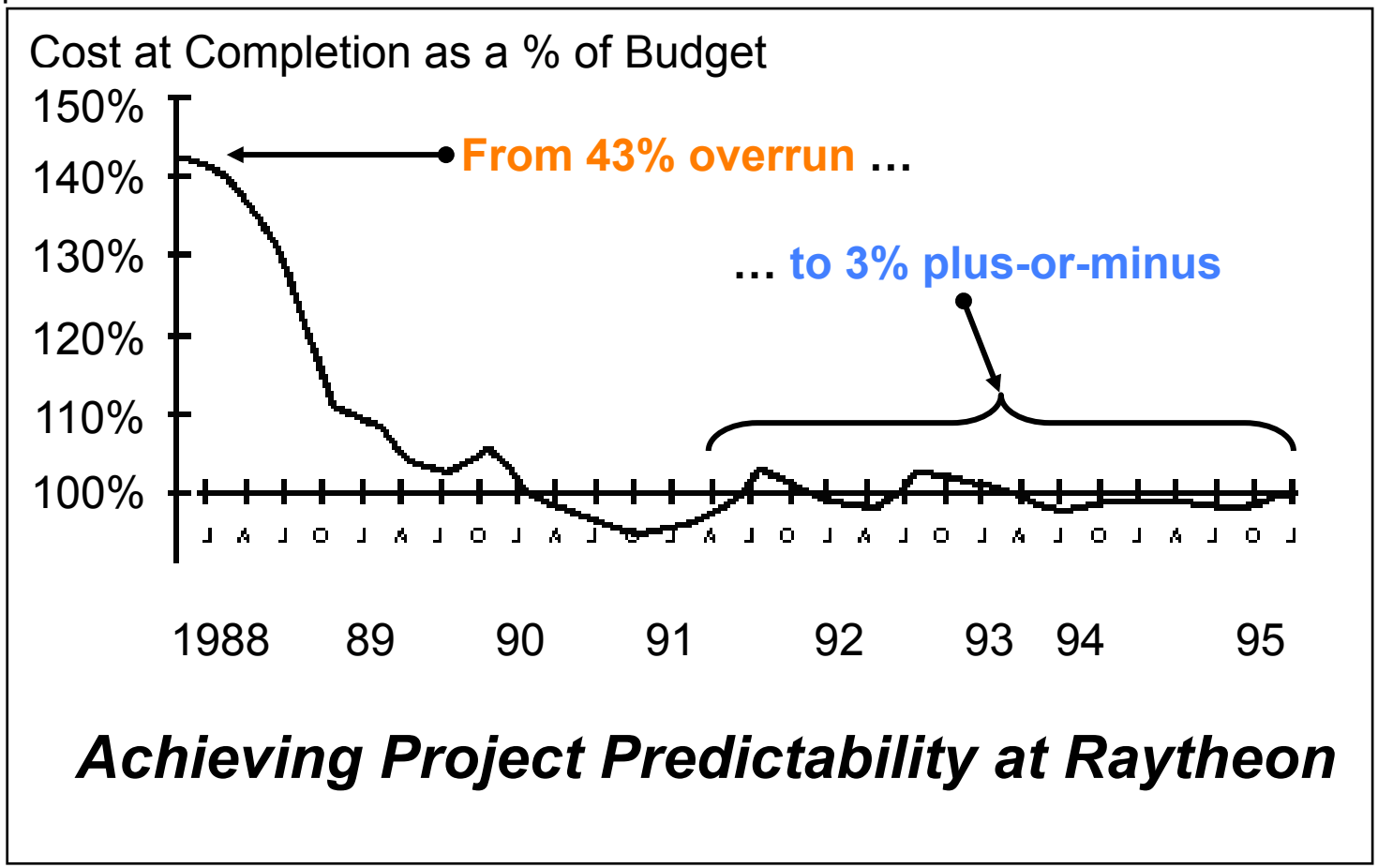
1. Performance requirements must be quantified!
2. Sources must be specified for all details!
3. Unambiguous to readership!
4. Clear enough to test!
5. Consistent with sources and siblings!

Competitiveness Rules. (Content)!

1. Number one in market performance levels!
2. Number one in cost levels!
3. Number one in service levels!
4. Number one in distribution capability!

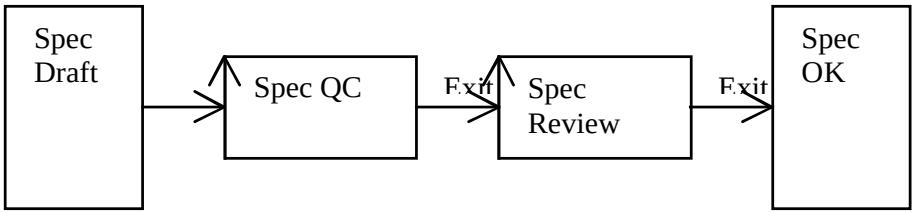
How does Planguage help Spec Quality Control?

- Planguage:
  - Provides specific standards to check for defects (rules, exit conditions, entry conditions)
  - Provides well defined and integrated processes for QC and all related processes of specification and project management
  - Contains structures which enable efficient cross checking of information by people and computers.
  - Contains a consistent set of standards and concepts for all types of specification - 'once learned applies to all'



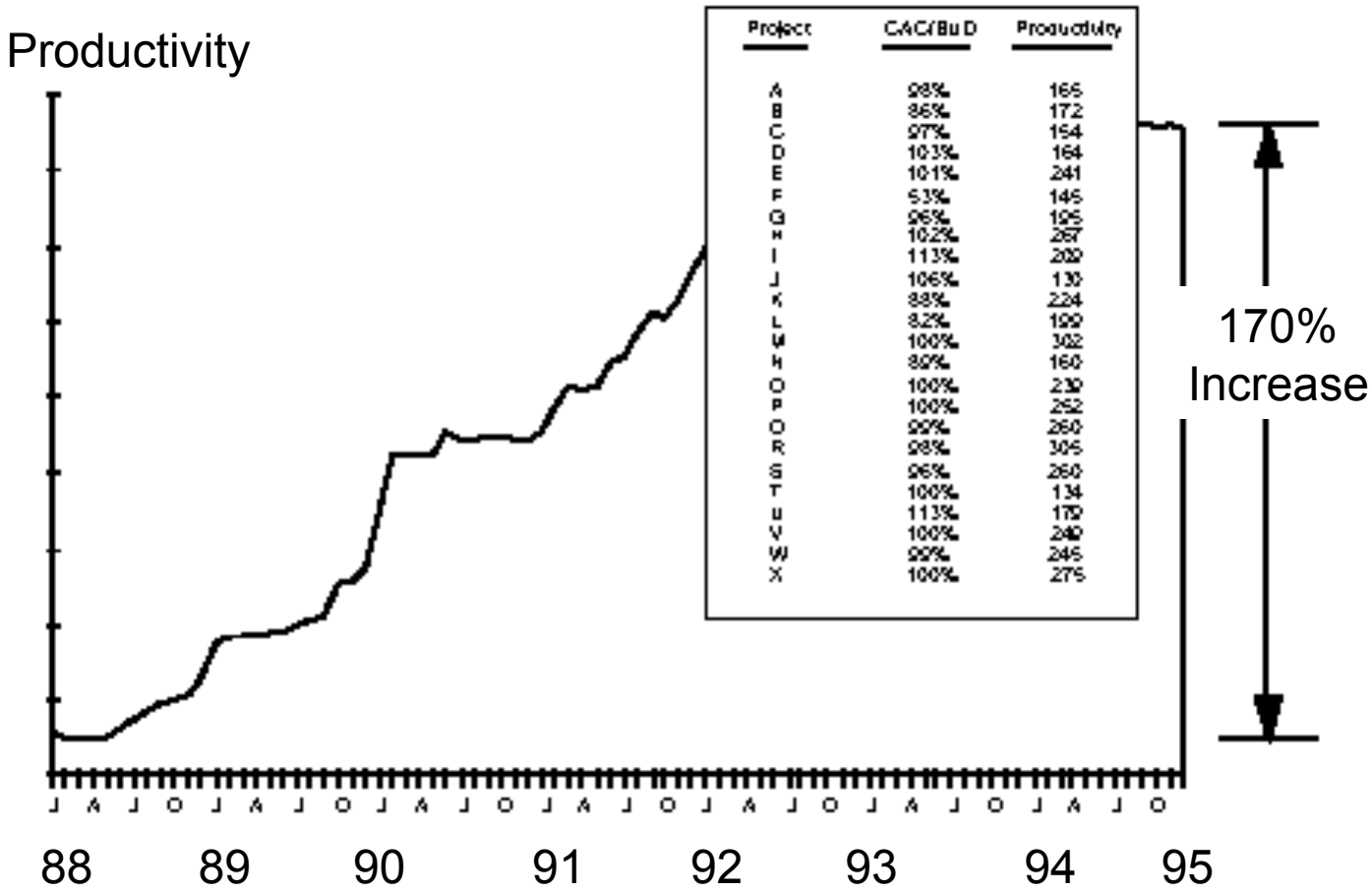
How does Planguage help Reviews?

- ! It ensures
  - –!*intelligible* and *consistent* specifications
  - –!Numeric exit from SQC *before* review
  - –!so that reviews are based on a solid foundation - and do not waste senior people's time, with sloppy work



How does Spec QC impact competitiveness?

- Indirectly
  - By avoiding rework (40%+ of total project cost if you are not careful!)
  - Speeds up projects by factor 2 to 3 (ex. Raytheon 95 SEI, below))



POSSIBLE PURPOSES FOR USING SQC

- Reducing Time-to-Delivery
- Measuring the Quality of a Document
- Measuring the Quality of the Process producing the Document
- Enabling Estimation of the Number of Remaining Defects
- Identifying Defects
- Removing Defects
- Preventing additional ‘Downstream’ Defects being generated by removing existing Defects
- Improving the Engineering Specification Process
- Improving the SQC Process
- On-the-Job Training for the Checkers
- Training the SQC Team Leader
- Certifying the SQC Team Leader
- Peer Motivation
- Motivating the Managers
- Helping the Specs Writer
- Reinforcing Conformance to Standards
- Capturing and Re-using Expert Knowledge (by use of Rules and Checklists)
- Reducing Costs
- Team Building
- Fun – a Social Occasion

Part 4: Impact Estimation Tables for quantified evaluation of design.

Finding competitive designs

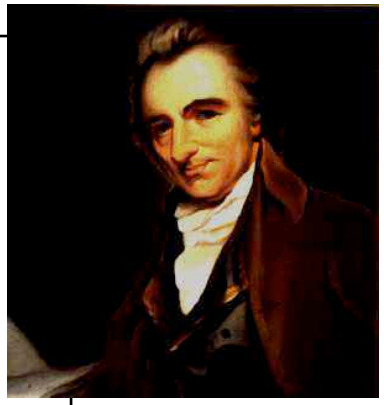
- What is a 'design'? (architecture, solution)
- What are the principles of evaluating a design?
- How do we evaluate a single dimension of impact?
- How can we evaluate all dimensions of impact?
- What uses can we put impact estimation to?
- How does Impact Estimation relate to Planguage?
- How do we specify a design with impacts?

Requirements \ Potential Design Solutions		Design Idea 1 'Standard' Pen	Design Idea 2 Laptop
Binary - Function Target	Recording Information	Yes	Yes
Function 1			
Binary - Design Constraint	Titanium Casing	No, Fail	Yes
Design Constraint 1			
Binary - Condition Constraint	Legal in the UK	Yes	Yes
Legal Constraint 1			
Scalar - Performance Target	Portability	20g	1Kg
Performance 1 →			
Scalar - Budget Target	Financial Cost	GBP 1	GBP 2.5K
Resource 1 →			

Evidence - by Thomas and John

"The most formidable weapon against errors of every kind is reason."

--Thomas Paine



"Facts are stubborn things; and whatever may be our wishes, our inclinations, or the dictates of our passions, they cannot alter the state of facts and evidence."

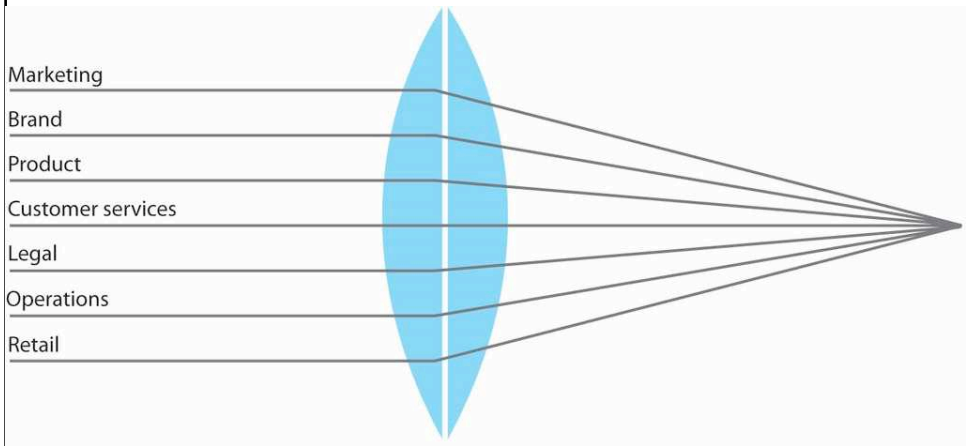
--John Adams

What is a ‘design’? (architecture, solution)

Design Idea!!

Concept *047 March 15, 2003 !

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



SCALAR REQUIREMENT SPECIFICATION

Participation: Scale: % of worldwide membership participating. Goal: 10%.

Representation: Scale: % of worldwide membership represented within defined <groups>.

Goal [Age under 25 or equating to <student status>]: 10%.

Information: Scale: % of talks rated as ‘good’ or better (5+ on feedback sheet scale). Goal: 50%.

Conviction: Scale: % participants wanting to return next conference. Goal: 80%.

Influence: Scale: % participants who <improve as result of the conference>.

Past: 90%, Goal: 95%.

Fun: Scale: % participants rating the conference-city quality as ‘good’ or better (5+ on feedback sheet scale).

Past: 45%. Plan: 60%.

Cost: Resource Budget: Scale: total cost for an individual participant including travel costs.

Fail: $2,000. Goal: $1,200 or less.

DESIGN SPECIFICATION (simple version)

Central: Choose a location in the membership center of gravity (New York?)

Youth: Suggest and support local campaigns to finance ‘sending’ a young representative to conference.

Facts: Review all submitted papers on <content>.

London: Announce that the conference is to be in London next time.

Diploma: Give diplomas for attendance, and additional diplomas for individual tutorial courses.

Events: Have entertainment activities organized every evening: river tours, etc.

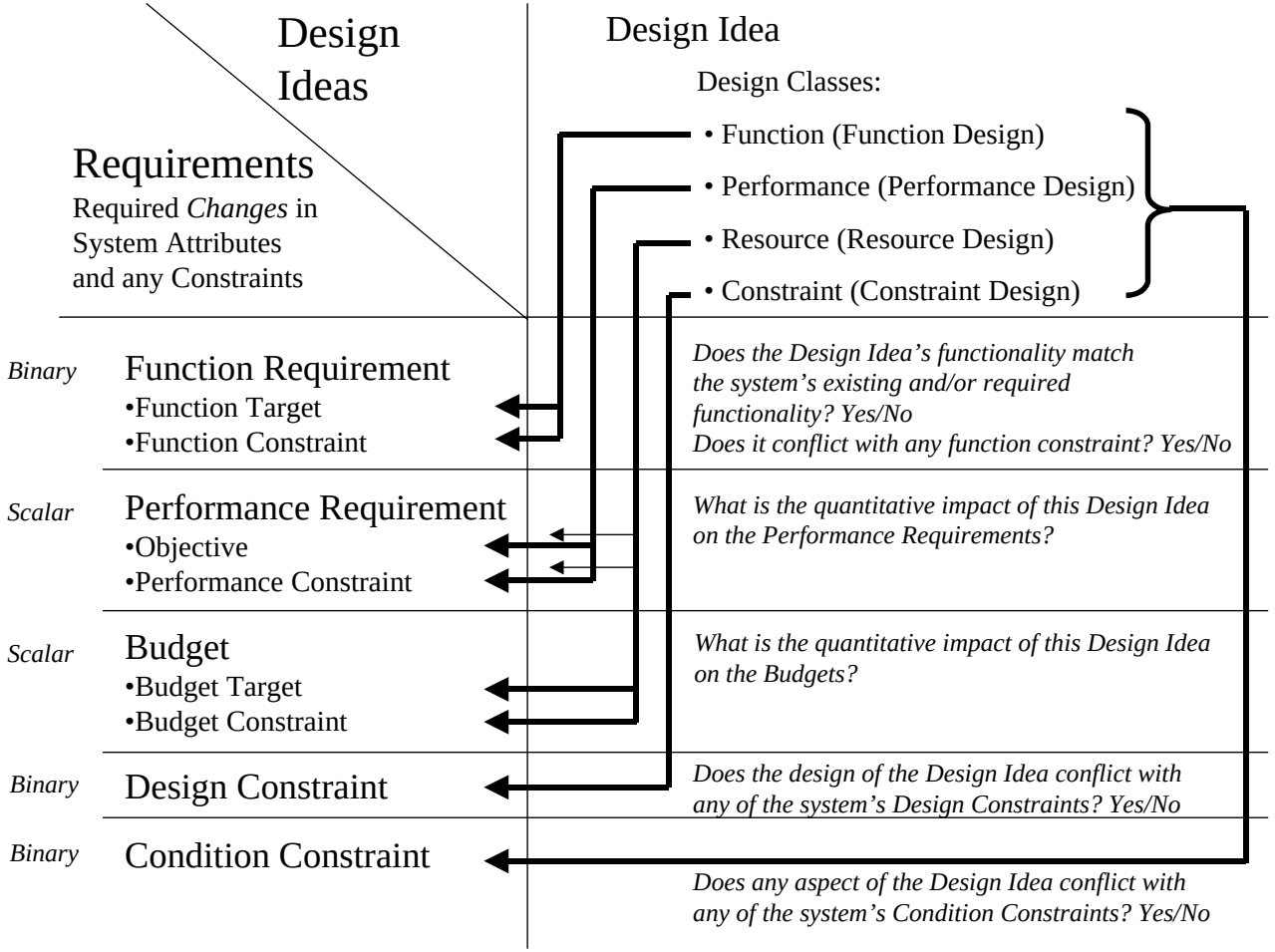
Discounts: Get discounts on airfare and hotels.

Example of a (Real, partial) Design Specification using Planguage

Tag: OPP Integration.
Type: Design Idea [Architectural].
========== Basic Information ==========
Version:
Status:
Quality Level:
Owner:
Expert:
Authority:
Source: System Specification Volume 1 Version 1.1, SIG, February 4, - Precise reference <to be supplied by Andy>.

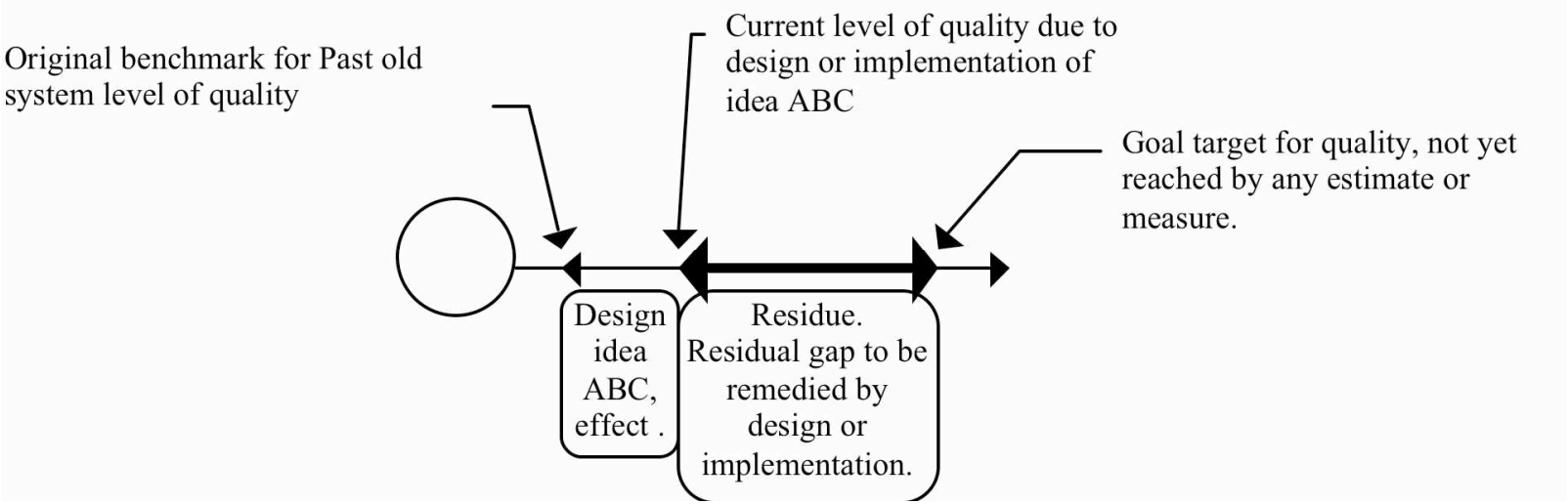
Gist: The X-999 would integrate both 'Push Server' and 'Push Client' roles of the Object Push Profile (OPP).
Description: Defined X-999 software acts in accordance with the <specification> defined for both the Push Server and Push Client roles of the Object Push Profile (OPP).
Only when official certification is actually and correctly granted; has the {developer or supplier or any real integrator, whoever it really is doing the integration} completed their task correctly.
This includes correct proven interface to any other related modules specified in the specification.
Stakeholders: Phonebook, Scheduler, Testers, <Product Architect>, Product Planner, Software Engineers, User Interface Designer, Project Team Leader, Company engineers, Developers from other Company product departments which we interface with, the supplier of the TTT, CC. "Other than Owner and Expert. The people we are writing this particular requirement for"
========== Design Relationships ==========
Reuse of Other Design:
Reuse of this Design:
Design Constraints:
Sub-Designs:
========== Impacts Relationships ==========
Impacts [Intended]: Interoperability.
Impacts [Side Effects]:
Impacts [Costs]:
Impacts [Other Designs]:
Value:
Interoperability: Defined As: Certified that this device can exchange information with any other device produced by this project.
========== Impact Estimation/Feedback ==========
Impact Percentage [Interoperability, Estimate]: <100% of Interoperability objective with other devices that support OPP on time is estimated to be the result>.
========== Priority and Risk Management ==========
Assumptions: There are some performance requirements within our certification process regarding probability of connection and transmission etc. that we do not remember <-TG.
Dependencies:
Risks: <none identified>.
We do not 'understand' fully (because we don't have information to hand here) our certification requirements, so we risk that our design will fail certification. <-TG
Priority:
Issues:
========== Location of Specification ==========
Location of Master Specification: <Give the Internet web location of the master specification>.

What are the principles of evaluating a design?



- ! Avoid violating constraints
- ! Meet Target and Function requirements

How do we evaluate a single dimension of impact?



- We must estimate or measure the numeric cumulative impact of the design
  - on a defined Scale,
  - using a defined Meter,
  - with respect to target and constraint levels.

How can we evaluate ***all*** dimensions of impact?

Design Ideas ***Objectives***	*Central*	*Youth*	*Facts*	*London*	*Diploma*	*Events*	*Discounts*	*Total*
Participation	*80%±50%*	*60%±70%*	*0%±50%*	*0%±50%*	*30%±50%*	*20%±50%*	*30%±50%*	*220%±370%*
Representation	*80%±50%*	*80%±50%*	*10%±50%*	*0%±50%*	*10%±50%*	*20%±50%*	*50%±40%*	*250%±340%*
Information	*0%±50%*	*20%±40%*	*80%±50%*	*0%±20%*	*20%±50%*	*0%±50%*	*0%±30%*	*120%±290%*
Conviction	*0%±10%*	*20%±50%*	*60%±30%*	*80%±50%*	*10%±50%*	*80%±50%*	*0%±50%*	*250%±290%*
Influence	*0%±50%*	*40%±40%*	*60%±50%*	*0%±50%*	*80%±50%*	*80%±50%*	*0%±50%*	*260%±340%*
Fun	*50%±50%*	*40%±50%*	*10%±50%*	*0%±0%*	*0%±0%*	*80%±50%*	*0%±0%*	*180%±200%*
Total	*210% ±260%*	*260% ±300%*	*220% ±280%*	*80% ±220%*	*150% ±250%*	*270% ±300%*	*80% ±220%*	
Budgets								
Cost	*10%*	*10%*	*10%*	*10%*	*1%±5%*	*50%±50%*	*80%±50%*	*171%±105%*
Benefit–to-Cost Ratio	*210%/10%*	*260%/10%*	*220%/10%*	*80%/10%*	*150/1*	*270/50*	*80/80*	

- ! We can use an Impact (Estimation) Table

What uses can we put impact estimation to?

IE can be used for a wide variety of purposes including:

1. **Evaluating a single design idea. How good is the idea for us?**
2. **Comparing two or more design ideas to find a winner, or set of winners. Use IE, if you want to set up an argument against a prevailing popular, but weak design idea!**
3. **Gaining an architectural overview of the impact of all the design ideas on all the objectives and budgets. Are there any negative side effects? What is the cumulative effect?**
4. **Obtaining systems engineering views of specific components, or specific performance aspects.**

Are we going to achieve the reliability levels?

5. **Analyzing risk: evaluating a design with regard to 'worst case' uncertainty and minimum credibility.**
6. **Planning evolutionary project delivery steps with regard to value and cost.**
7. **Monitoring, for project management accounting purposes, the progress of individual evolutionary project delivery steps and, the progress to date compared against the requirement specification or management objectives.**
8. **Predicting future costs, project timescales and performance levels.**
9. **Understanding organizational responsibility in terms of performance and budgets by organizational function.**

   In 1992, Steve Poppe pioneered this use at executive level while at British Telecom, North America.

10. **Achieving rigorous quality control of a design specification prior to management reviews and approval.**
11. **Presenting ideas to committees, management boards, senior managers, review boards and customers for approval.**
12. **Identifying which parts of the design are the weakest (risk analysis). If there are no obvious alternative design ideas, any 'weak links' should be tried out earliest, in case they do not work well (risk management). This impacts scheduling.**
13. **Enabling configuration management of design, design changes, and change consequences.**
14. **Permitting delegation of decision-making to teams. Teams can achieve better internal progress control using IE, than they can from repeatedly making progress reports to others, and acting on others' feedback.**
15. **Presenting overviews of very large, complex projects and systems by using hierarchical IE tables. Aim for a one page top-level IE view for senior management.**
16. **Enabling cross-organizational co-operation by presenting overviews of how the design ideas of different projects contribute towards corporate objectives. Any common and conflicting design ideas can be identified. This is important from a customer viewpoint; different projects might well be delivering to the same customer interface.**
17. **Controlling the design process. You can see what you need, and see if your idea has it by using an IE table. For example, which design idea contributes best to achieving usability? Which one costs too much?**
18. **Strengthening design. You can see where your design ideas are failing to impact sufficiently on the objectives; and this can provoke thought to discover new design ideas or modify existing ones.**
19. **Helping informal reasoning and discussion of ideas by providing a framework model in our minds of how the design is connected to the requirements.**
20. **Strengthening the specified requirements. Sometimes, you can identify a design idea, that has a great deal of popular support, but doesn't appear to impact your requirements. You should investigate the likely impacts of the design idea with a view to identifying additional stakeholder requirements. This may provide the underlying reason for the popular support. You might also identify additional types of stakeholders.**

Deeper Into Estimation Parameters?

Learning:

Ambition: Make it substantially easier for our users to learn tasks <- Marketing.

Scale: Average time for a defined [User Type: default UK telesales trainee] to learn a defined [User Task: default Response] using <our product's instructional aids>.

Response: Task: Give correct answer to simple request.

Past [last year]: 60 minutes.

GN: Goal [By start of next year]: 20 minutes.

GA: Goal [By start of year after next]: 10 minutes.

	On-line Support	On-line Help		Picture Handbook	On-line Help + Access Index
Learning Past: 60min. <<-> Plan: 10min.					
Scale Impact	5 min.	10 min.		30 min.	8 min.
Scale Uncertainty	±3min.	±5 min.		±10min.	±5 min.
Percentage Impact	110%	100%		67% (2/3)	104%
Percentage Uncertainty	±6% (3 of 50 minutes)	±10%		±20%?	±10%
Evidence	Project Ajax, 1996, 7 min.	Other Systems		Guess	Other Systems + Guess
Source	Ajax report, p.6	World Report p.17		John B.	World Report p.17 + John B.
Credibility	0.7	0.8		0.2	0.6
Development Cost	120K	25K		10K	26K
Benefit-To-Cost Ratio	110/120 = 0.92	100/25 = 4.0		67/10 = 6.7	104/26 = 4.0
Credibility-adjusted B/C Ratio (to 1 decimal place)	0.92*0.7 = 0.6	4.0*0.8 = 3.2		6.7*0.2 = 1.3	4.0*0.6 = 2.4
Notes: Time Period is two years.	Longer timescale to develop				

Picture Handbook: Gist: Produce a radically changed handbook that uses pictures and concrete ! examples to *instruct*, without the need for *any* other text. **!**

Real (NON-CONFIDENTIAL version) example of an initial draft of setting the objectives that engineering processes must meet.

Business objective	Measure	Goal (200X)	Stretch goal ('0X)	Volume	Value	Profit	Cash
Time to market	Normal project time from GT to GT5	<9 mc	<6 mc	X		X	X
Mid-range	Min BoM for The Corp phone	<$9	3	X		X	X
Platformisation Technology	# of Technology 66 Lic. shipping > 3M/yr	4	6	X		X	X
Interface	Interface units	>11M	>13M	X		X	X
Operator preference	Top-3 operators issue RFQ spec The Corp		2	X		X	X
Productivity						X	X
Get Torden	Lyn goes for Technology 66 in Sep-04	Yes		X		X	X
Fragmentation	Share of components modified	<10%	<5%		X	X	X
Commoditisation	Switching cost for a UI to another System	>1y	>2 yrs			X	X
Duplication	The Corp share of 'in scope' code in best-selling device	>90%	>95%		X	X	X
Competitiveness	Major feature comparison with MX	Same	Better	X		X	X
User experience	Key use cases superior vs. competition	5	10	X	X	X	X
Downstream cost saving	Project ROI for Licensees	>33%	>66%	X	X	X	X
Platformisation IFace	Number of shipping Lic.	33	55	X		X	X
Japan	Share of of XXXX sales	>50%	>60%	X		X	X
	Numbers are intentionally changed from real ones						

Business Objectives Quantified

Strategy Impact Estimation:
for a $100,000,000 Organizational Improvement Investment

Objectives

Technical Strategies

"Benefits"

Strategy Impacts on Objectives

Cost

Benefit/Cost ratio

358 !

Business Objective		hardware adaptation	Telephony	Reference designs	IFace	Modularity	Defend vs Technology 66	Tools	User Experience	GUI & Graphics	Security	Defend vs OCD	Enterprise
		Viking Deliverables											
Time to market		20%	10%	30%	5%	10%	5%	15%	0%	0%	0%	5%	5%
Mid-range		15%	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	5%	10%	5%	5%	0%	0%
Platformisation Technology		25%	10%	30%	0%	[illegible]	10%	0%	5%	0%	10%	0%	5%
Interface		5%	15%	15%	0%	5%	0%	5%	0%	0%	10%	0%	10%
Operator preference		0%	[illegible]	[illegible]	[illegible]	[illegible]	20%	5%	10%	10%	20%	5%	10%
Get Torden		25%	10%	10%	-10%	0%	20%	0%	10%	-20%	10%	10%	5%
Commoditisation		20%	10%	20%	10%	-20%	25%	15%	0%	0%	5%	10%	5%
Duplication		15%	[illegible]	10%	0%	0%	40%	0%	0%	0%	5%	20%	5%
Competitiveness		10%	15%	20%	0%	10%	20%	10%	10%	20%	10%	10%	10%
User experience		5%	[illegible]	0%	0%	[illegible]	0%	0%	30%	10%	0%	0%	0%
Downstream cost saving		15%	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	10%	0%	0%	10%	5%
Platformisation IFace		10%	10%	20%	40%	0%	20%	5%	0%	0%	0%	0%	5%
Japan		10%	5%	20%	0%	[illegible]	0%	0%	[illegible]	5%	0%	0%	0%
Contribution to overall result		15%	9%	17%	4%	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	5%
Cost (£M)	£	2.85	£ 0.49	£ 3.21	£ 2.54	£ 1.92	£ 2.31	£ 0.81	£ 1.21	£ 2.68	£ 0.79	£ 0.62	£ 0.60
ROI Index (100=average)		106	358	[illegible]	33	78	[illegible]	148	107	10	152	202	174

Nordic Road Building Software IE:
Selecting the most *competitive* investments

	Road Design Functions						Road Data Model		Drawing Production		
	Road Standard (Requirements)	Road Network	Alignment Design	Road modelling	Intersection modelling (3D!)	Analyse the Design	Storage of road model	Storage of Alignments	Drawing Functions	Drawing Factory	CAD Compor
Product Qualities											
Efficiency.Design,	5%	30%	20%	40%	15%	20%	10%	15%	30%	20%	0%
Efficiency.Construction	0%	5%	0%	40%	20%	10%	10%	0%	0%	0%	0%
Efficiency. Facility management	0%	20%	0%	10%	5%	0%	10%	10%	0%	0%	0%
Efficient.Localisation	-20%	0%	0%	0%	15%	-5%	10%	0%	30%	20%	0%
Quality.Localisation	-20%	0%	0%	0%	0%	0%	10%	0%	20%	15%	0%
Usability.Learnability	0%	10%	30%	30%	15%	-5%	5%	10%	10%	10%	0%
Usability.Intuitive	-5%	10%	20%	30%	15%	-5%	10%	10%	10%	10%	0%
Usability.Fun	10%	10%	20%	20%	10%	5%	5%	0%	15%	15%	0%
Usability.Workflow	20%	40%	10%	20%	15%	0%	5%	10%	10%	10%	0%
Availability.Reliability	0%	-10%	-10%	-10%	-10%	0%	10%	0%	5%	5%	0%
Availability.Maintainability	0%	-10%	-10%	-10%	-10%	0%	10%	0%	5%	5%	0%
Availability.Scaleability	0%	-10%	-10%	-10%	20%	0%	20%	0%	10%	10%	0%
Portability	0%	0%	0%	0%	20%	0%	15%	10%	10%	10%	0%
Identity. Novapoint	30%	30%	30%	0%	10%	15%	30%	10%	5%	5%	0%
	20%	**125%**	**100%**	**160%**	**140%**	**35%**	**160%**	**75%**	**160%**	**135%**	**0%**
Engineers.Innhouse											
15,000	300	1000	80	1000	1000	100	2500	100	0		
Engineers.External											
Thai	300								1000		
Vietnam						300					
Partners		300	200		1000			80			
Sweden										800	
Denmark											
Finland											
Others											
Total Development Resources	600	1300	280	1000	2000	400	2500	180	1000	800	
Benefit / Dev. Resources	0.03%	0.10%	**0.36%**	**0.16%**	0.07%	0.09%	0.06%	**0.42%**	**0.16%**	**0.17%**	0
			2	3				1	3	4	

How do we specify a design with *impacts?*
A Template to make us think competitively

Tag: <Unique Name Capitalized>

Type: Design Idea.

Version: <date and or version number of last change>

Owner: < originator, champion, expert, maintainer, architect, systems engineer>

Description: <describe the design in a dozen, or more, words. The detail should be sufficient to guarantee the expected impacts and costs estimated below>.

Reuse: <if a currently available component or design is specified, then give it's tag or reference code here to indicate that a known component is being applied>

Primary Impacts: <give the main impact or impacts which this design is expected to have on an objective . Thes are its main justification for existence!>.

Secondary Impacts: <list expected secondary impacts, good or bad>.

Cost Impacts: <give at least rough impacts on defined budget constraints>.

========== More Formal Impact Estimation ==========

Real Impact on defined Scale: <give expected impact result on the Scale defined, when implemented>

%Impact on Specific Goal: <Convert real impact to % impact relative to the main planned level: 100% means meets defined Plan level on time>.

± %Uncertainty: <give optimistic/pessimistic % deviation, like ±20%, based on best and worst real observations

Evidence: <give the observed numbers, facts, dates, places where you have data about this designs impact>

Source: <give the person or written source of your evidence>

Credibility: <Credibility 0.0 low to 1.0 high. Rate the quality of your estimates, based on the historic data you have>

--------- Repeat this sequence for any other major impact objectives you believe justify the specification effort here.

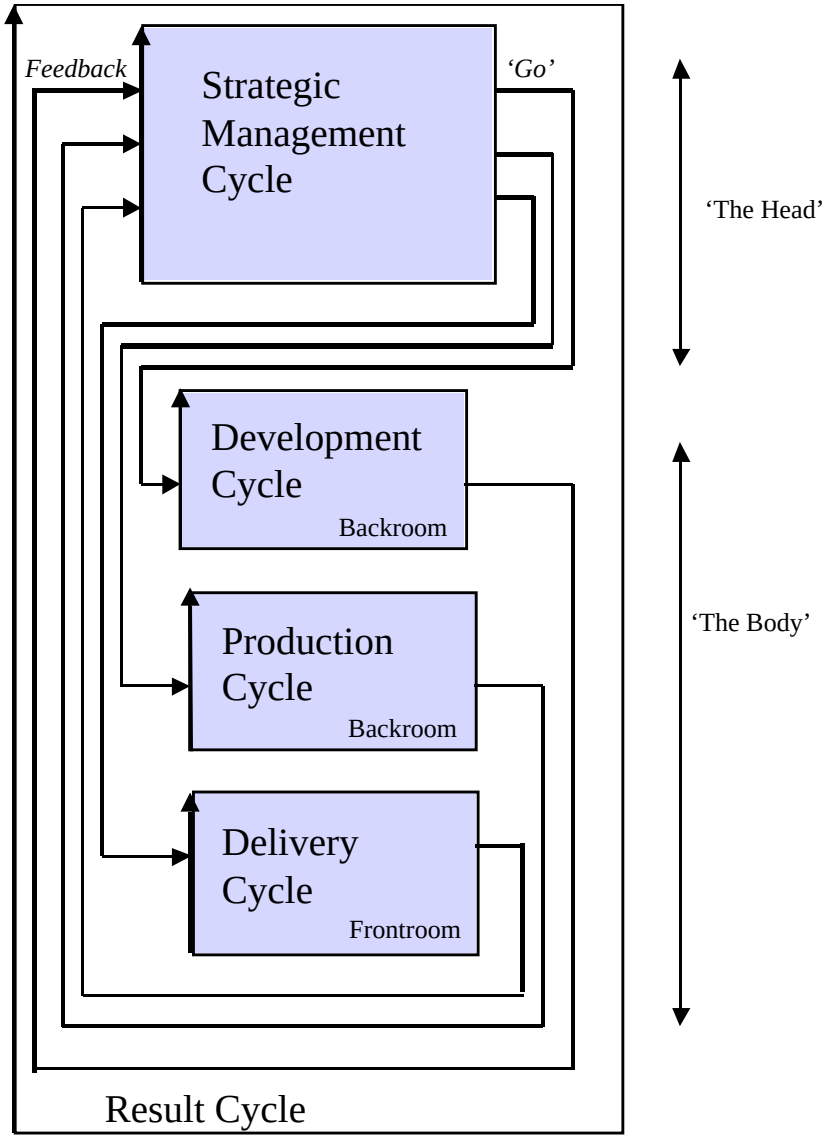
Expert: < name and give contact (email?) a useful technical expert in our company or otherwise available to us on this design idea>.

Authority: <name and give contact information to the leading authorities in our co. or elsewhere on this technology. Reference papers or books for example and websites>

Web Location of Master Specification: <give intranet web location of this master specification>.

Part 5: Evolutionary Project Management

- The fundamentals of an Evo step
- How does Planguage support Evo project management?
- How do you plan an Evo step in Planguage?
- How does Evo relate to requirements?
- How does Evo relate to Design?
- How does Evo relate to Risk?
- How does Evo relate to process improvement?
- How does Evo relate to competitiveness?



confirmit everywhere

Evo in Confirmit

Presented by:
Trond Johansen
Software Development Manager

confirmit everywhere

Presentation overview

- ✓! Evo in short
- ✓! Evolutionary project management
- ✓! Requirements
- ✓! Designs &Solutions
- ✓! Evo planning, IET, FIRM Evo cycle
- ✓! Evo's impact on Confirmit product qualities
- ✓! Benefits of Evo for clients

Characteristics of Evo

- Evo is characterized by:
  - Focus on quantified stakeholder values and product qualities
    - Features & functionality comes as a result of these
  - Frequent deliveries, two-weeks development cycle
  - Frequent feedback from stakeholders
  - Measurements and metrics – Numbers can provide evidence of whether we are heading in the right direction with respect to the product qualities.

Method developed by Tom Gilb (www.gilb.com) and applied by Nokia, Intel, Microsoft, Ericsson, Sun Microsystems, Phillips, HP etc

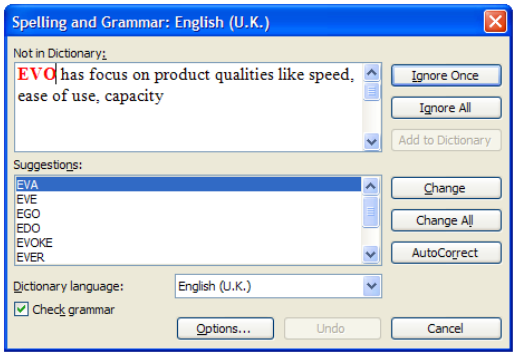
Overview of Evo

- ✓! Find stakeholders (End users, super-users, support department, IT operations, marketing etc) – focus on the most important ones
- ✓! Define the stakeholders real needs and what product qualities that can fulfill these needs.
- ✓! Identify past/status of product qualities and your goal (how much you want to improve).
- ✓! Identify possible designs/solutions for meeting your goals
- ✓! Develop a step-by-step plan for delivering, not solutions, but improvements to Stakeholder Values & Product Quality goals.
  - –! Deliveries every second week!
  - –! Measure: are we moving towards our goals?

A comprehensive description of the method can be found in
"Competitive Engineering" by Tom Gilb

Requirement management in Evo

- ✓! Evo is different from other standard requirement processes which mostly focus on function requirements. Evo focus on product quality requirements, because it is the quality requirements that separate one product from another.
- ✓! Example: Consider a spell checker in word and a paper based dictionary, which one do you prefer, and why? The core feature set is pretty much the same, checkiiiiinggng your spelling..
  - –! Superior product qualities: Performance.Speed, Usability



Defining requirements

- We try to define our requirements according to a basic standard (in "Competitive Engineering", Rules by Tom Gilb):
  - Clear & Unambiguous
  - Testable
  - Measurable
  - No Solutions/designs. How often haven't we seen statements like this:"The screen must contain a button that does x y z", instead of focusing on the workflow they are trying to optimize
  - Stakeholder Focus
    - The ones that pay for the product: productivity, scalability, performance
    - The ones that use the system: Usability, intuitiveness

Product quality - example

- **Usability.Productivity**
  - **Scale: Time in minutes to set up a typical specified MR-report (what to measure)**
  - **Past: 65 min, Tolerable: 35 min, Goal: 25 min**
  - **(end result was 20 min ☺)**
  - **Meter: Candidates with Reportal experience and with knowledge of MR-specific reporting features performed a set of predefined steps to produce a standard MR Report (how to measure)**

- The focus is on the day-to-day operations of our users, not a list of features that they might or might not like. We know that increased efficiency will be appreciated!

Design Ideas

- For every quality requirement we look for possible Design Ideas
- E.g. for Quality Requirement: Usability.Productivity we identified the following Design Ideas:
  - DesignIdea.**Recoding** **Estimated Impact** **20 Minutes**
  - DesignIdea.MRTotals 13
  - DesignIdea.Categorizations 8
  - DesignIdea.TripleS 3
  - ..and many more
- We evaluated all these, and specified in more detail those we believed would add the most value (take us closer to the goal)
- A chosen Design Idea = Solution

Solutions

- A Solution is defined as a code change with the intention of improving a product quality. Such code changes are in most cases new features, but it can also be tuning of existing code. A Solution can also be implementation of a core functional requirement.
- A Solution is a work item with defined attributes. The most important attributes for a Solution is:
  - Summary: WHAT the solution does
  - Rationale: WHY this is a smart thing to do
  - A description of what the Solution consist of. It should be detailed enough for your peer to understand.
    - GUI tasks (UI components: new screens, buttons etc)
    - Database tasks (new tables, columns etc)
    - New classes, methods etc
    - Tests (Automated and manual)

Evo planning

- ✓! We collect the most promising and include them in an Evo plan (also called Impact Estimation Table: IET)
- ✓! The IET is our tool for controlling the qualities and deliver improvements to real stakeholders, or as close as we can get to them. (e.g. Our own support department acting as clients)
- ✓! One Evo step = 2 weeks!

Evo planning - example

- ! IET for MR Project – Confirmit 8.5
- ! **Solution:** Recoding
  - ! Make it possible to recode variable on the fly from Reportal.
  - ! Estimated effort: 4 days

	A	B	C	D	E	F	G	BX	BY	BZ	CA
1											
2		Current Status	Improvements		Goals			Step9			
3								Recoding			
4								Estimated impact		Actual impact	
5		Units	Units	%	Past	Tolerable	Goal	Units	%	Units	%
6					Usability.Replacability (feature count)						
7		1,00	1,0	50,0	2	1	0				
8					Usability.Speed.NewFeaturesImpact (%)						
9		5,00	5,0	100,0	0	15	5				
10		10,00	10,0	200,0	0	15	5				
11		0,00	0,0	0,0	0	30	10				
12					Usability.Intuitiveness (%)						
13		0,00	0,0	0,0	0	60	80				
14					Usability.Productivity (minutes)						
15		20,00	45,0	112,5	65	35	25	20,00	50,00	38,00	95,00
20					Development resources						
21			101,0	91,8	0		110	4,00	3,64	4,00	3,64

Product quality versus code quality

- ✓! Evo is focusing on delivering improvements to product qualities
- ✓! These **product qualities** materialize themselves as designs/solutions, often as new features/functionality
- ✓! To control the **code quality** of these new features we have put together a simple checklist in our IET framework

Evo planning – value vs. cost

- Project management meetings
  - In the project management meetings, each project leader present the results from the previous step (IET) as well as the content of next Evo step (one week)
  - Possible new Solutions are discussed and weighted against each other: **Most value for development resources**

confirmit everywhere

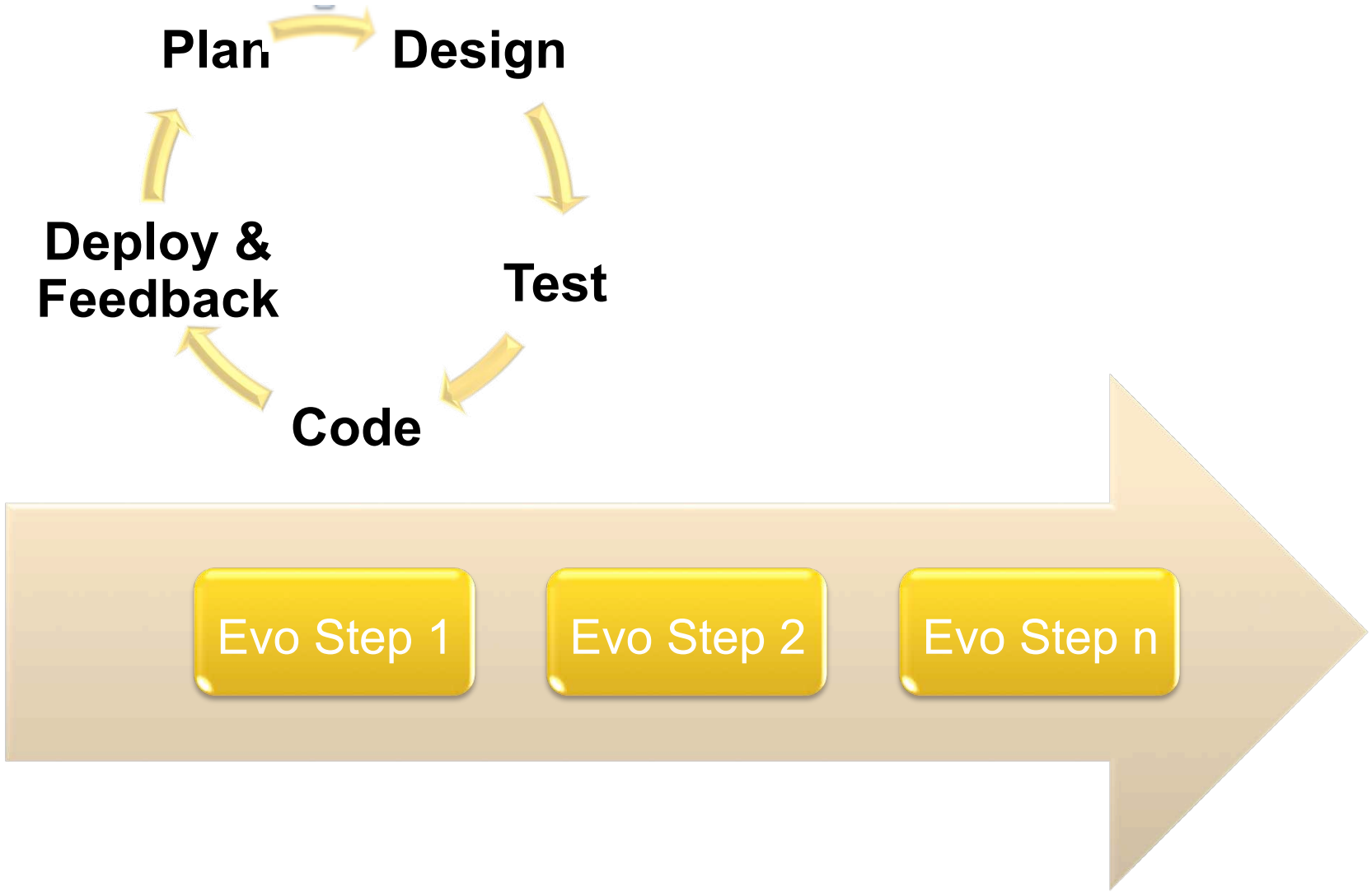
From concepts to day to day operations

- Confirmit's Evo implementation has the following attributes
  - Product Qualities
  - Design Ideas
  - Solutions
  - Evo Step
  - IET
  - Project Management Meetings
  - Design Review Meetings

- How are these connected in order to form our Evo development process?

Evo cycles

Friday	Feature team & Project Management Meeting: Review the quality of last Evo step and discuss design ideas for next step.
Monday	Write detailed Solutions and present them in design review meeting. Short debrief meeting with project team
Tuesday - Friday	Development
Monday	Development & Get feedback from all stakeholders. Timing can be adjusted by the project
Tuesday	Development
Wednesday	Development, finalize Evo step
Thursday	Feature team (Maintenance) and project planning


Plan
Design
Test
Code
Deploy &
Feedback
Evo Step 1
Evo Step 2
Evo Step n

confirmit everywhere

Evo's impact on Confirmit 8.5 product qualities: Top 5

Product quality	Past	End state
Usability.Productivity: Time for the system to generate a defined complex survey	**7200 secs**	**15 secs**
Usability.Productivity: Time to set up a typical specified Market Research report	**65 min**	**20 min**
Usability.Productivity: Time to grant a set of end-users access to a report set and distribute report login info	**80 min**	**5 min**
Usability.Intuitiveness: The time it takes a medium experienced programmer to create a complete and correct data transfer definition with Confirmit web services without any user documentation or other aid	**15 min**	**5 min**
Performance.Runtime.Concurrency: Maximum number of simultaneously respondents executing a survey with a click rate of 20 seconds and a response time <500 ms given a defined [Survey complexity] and a defined [Server configuration, Typical]	**250 users**	**6000 users**

COMPETITIVE RESULTS: Large, rapid and regular improvement in user-appreciated attributes

Evo's impact on Confirmit 9.0 product qualities

confirmit everywhere

Product quality	Customer value
Intuitiveness: Probability that an inexperienced user can intuitively figure out how to set up a defined Simple Survey correctly	Probability increased by **175**%
Productivity: Time in minutes for a defined advanced user, with full knowledge of 9.0 functionality, to set up a defined advanced survey correctly	Time reduced by **38**%
Productivity: Time (in minutes) to test a defined survey and identify 4 inserted script errors, starting from when the questionnaire is finished to the time testing is complete and is vey: Complex survey, 60 ng.)	Time reduced by **83**% and error tracking increased by **25**%

← **Intuitiveness!!**

COMPETITIVE RESULTS: Large, rapid and regular improvement in user-appreciated attributes

confirmit everywhere

Evo's impact on Confirmit 9.0 (2nd Quarter) product qualities

Product quality!	Description!	Customer value !
Performance!	Max number of panelists that the system can support without exceeding a defined time for the defined task, with all components of the panel system performing acceptable.!	Number of panelists increased by **1500**% !
Scalability!	Ability to accomplish a bulk-update of X panelists within a timeframe of Z second! !	Number of panelists increased by **700**%!
Performance!	Number of responses a database can contain if the generation of a defined table should be run in 5 seconds.!	Number of responses increased by **1400**%!

COMPETITIVE RESULTS: Large, rapid and regular improvement in user-appreciated attributes

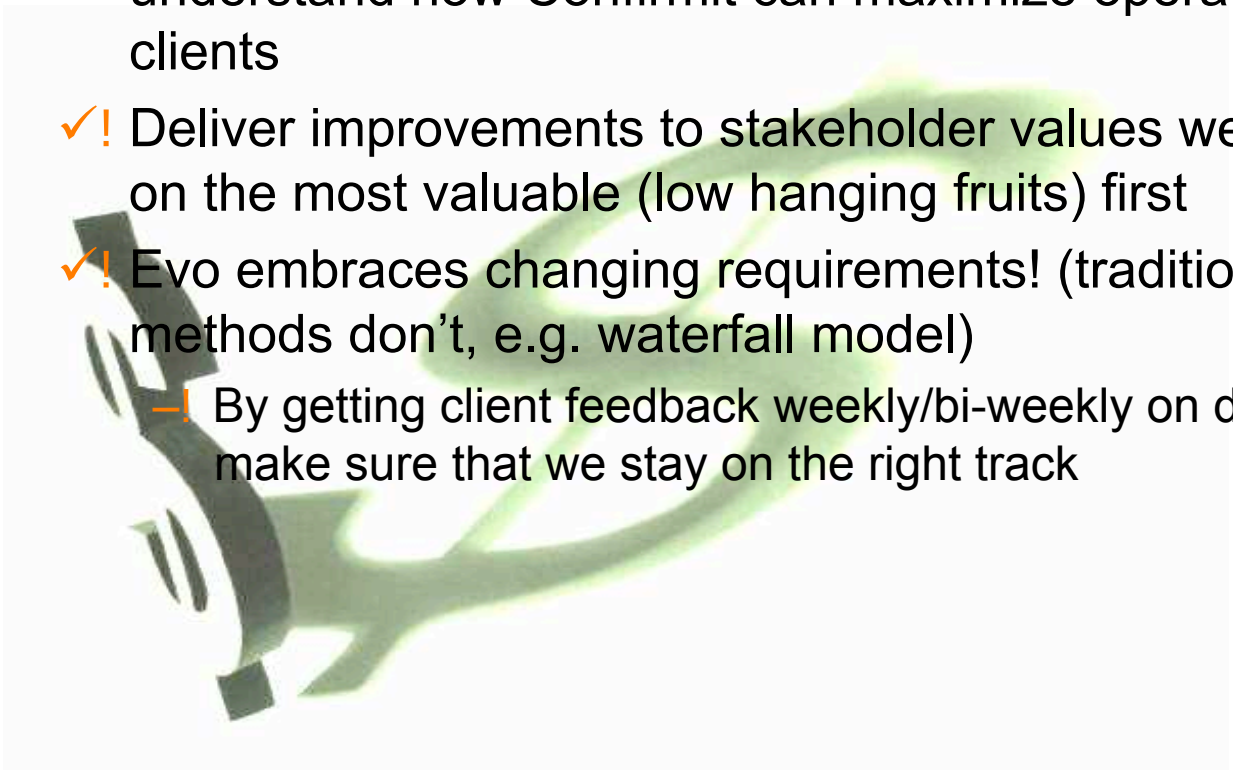
Evo as a tool for prioritization

- One of the strengths of Evo is the method's power of **focusing on delivering value** for clients **versus cost** of implementation.

- Evo enables us to **re-prioritize the next development-steps** based on **weekly feedback from our stakeholders**
  - What seemed important at the start of the project may be replaced by other solutions based on gained knowledge from previous steps.

Benefits of Evo for clients

- Identifying REAL stakeholder values in order for Confirmit to understand how Confirmit can maximize operating efficiency for the clients
- Deliver improvements to stakeholder values week by week, focusing on the most valuable (low hanging fruits) first
- Evo embraces changing requirements! (traditional development methods don't, e.g. waterfall model)
  - By getting client feedback weekly/bi-weekly on developed functionality we make sure that we stay on the right track



confirmit everywhere

Green Week: Improving Maintainability 1 week/month

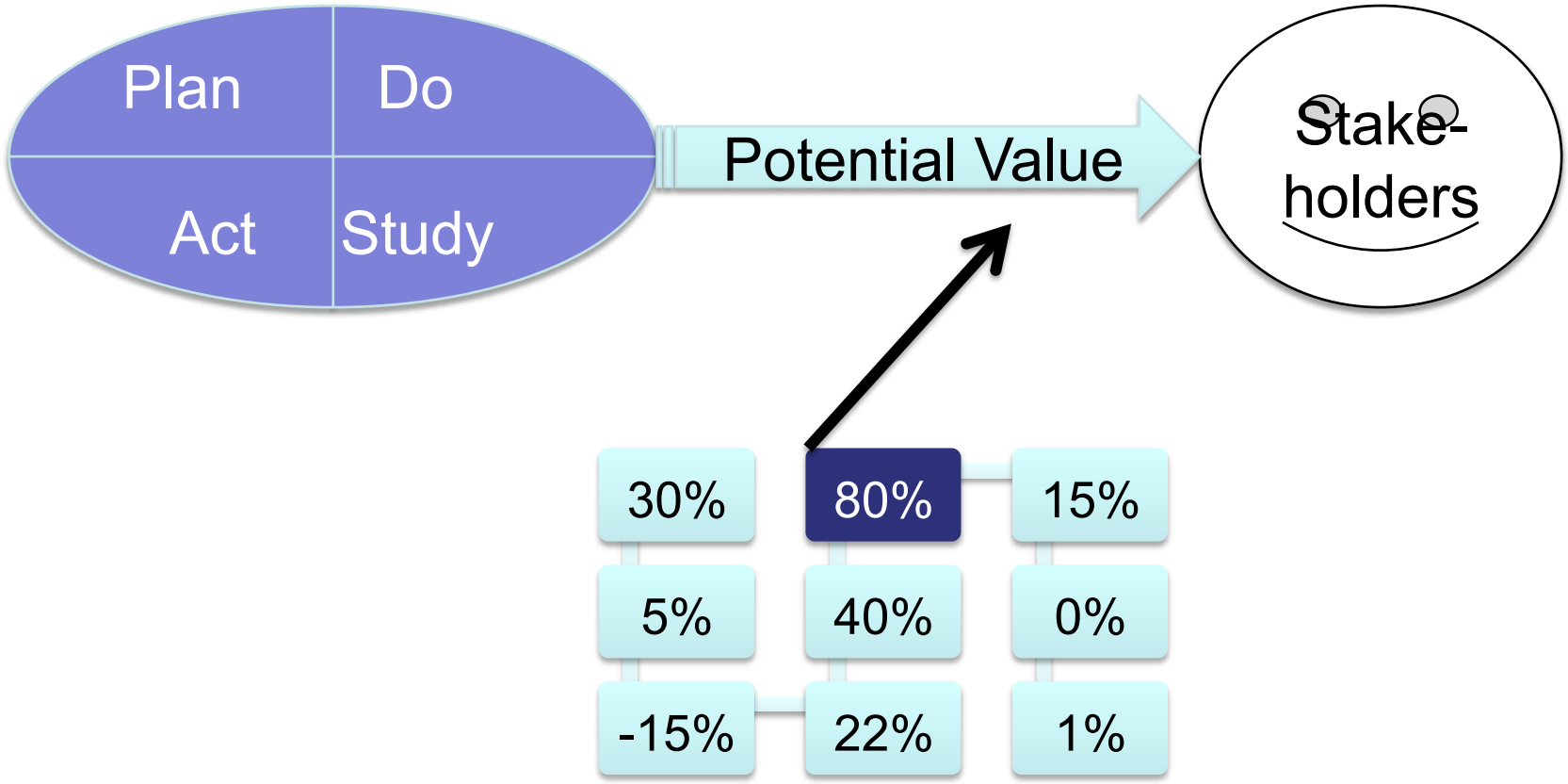
Current Status		Improvement	Goals			Step 6 (week 14)		Step 7 (week 15)	
	Units		Past	Tolerable	Goal	Estimated Impact	Actual Impact	Estimated Impact	Actual Impact
	100,0	100,0	0	80	100			100	100
Speed									
	100,0	100,0	0	80	100	100	100		
Maintainability.Doc.Code									
	100,0	100,0	0	80	100	100	100		
InterviewerConsole									
NUnitTests									
	0,0	0,0	0	90	100				
PeerTests									
	100,0	100,0	0	90	100			100	100
FxCop									
	0,0	10,0	10	0	0				
TestDirectorTests									
	100,0	100,0	0	90	100			100	100
Robustness.Correctness									
	2,0	2,0	0	1	2	2	2		
Robustness.BoundaryConditions									
	0,0	0,0	0	80	100				
Speed									
	0,0	0,0	0	80	100				
ResourceUsage.CPU									
	100,0	0,0	100	80	70	70			
Maintainability.Doc.Code									
	100,0	100,0	0	80	100	100	100		
SynchronizationStatus									
NUnitTests									

Primary Evo Concept: Deliver *Potential* Value

The Evo Cycle:
Viewed as a Deming PDSA Cycle

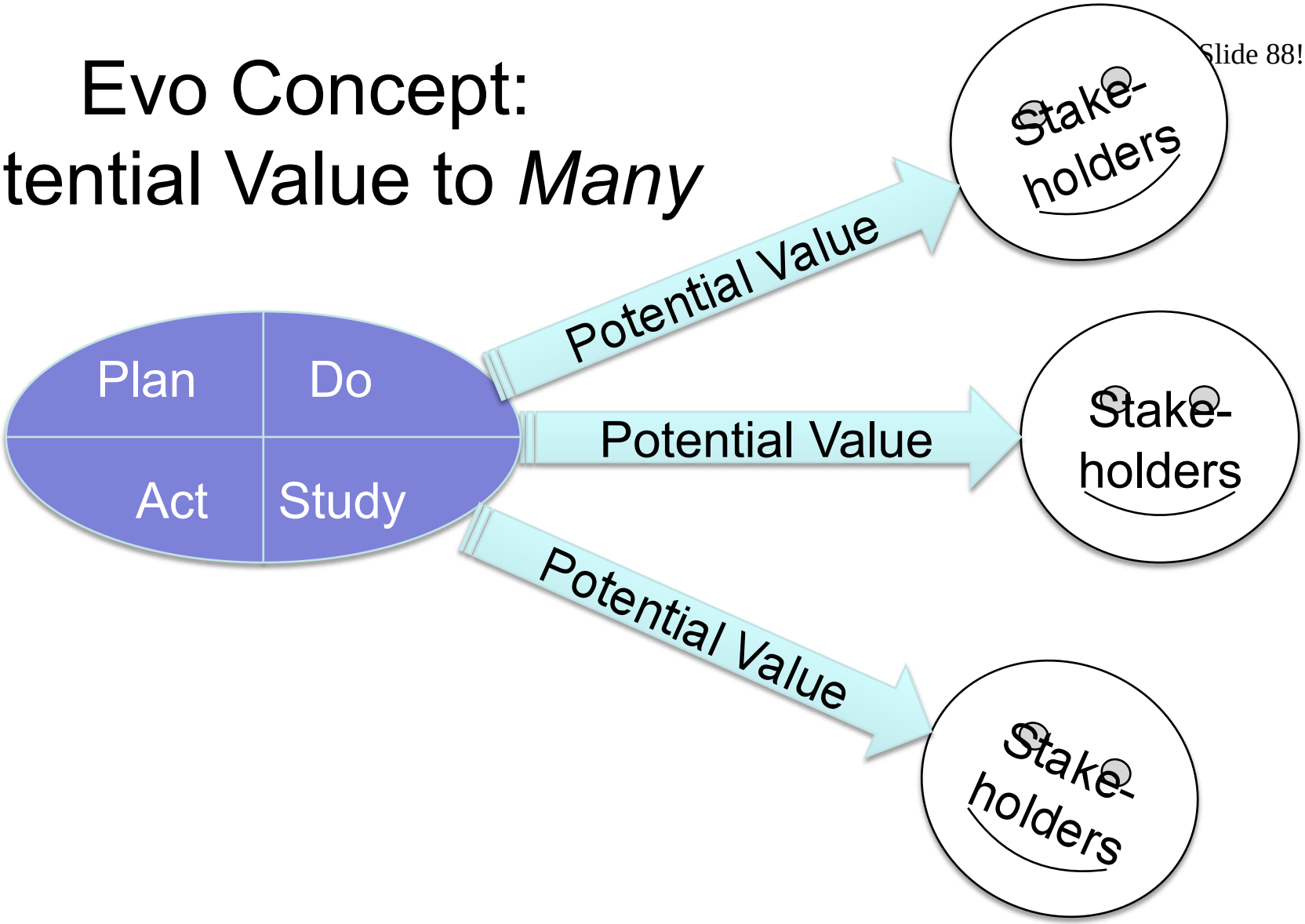
•! Incremental Value Delivery to Stakeholders

Deliver the highest value for resources



HIGHEST AVAILABLE Incremental Value Delivery to Stakeholders

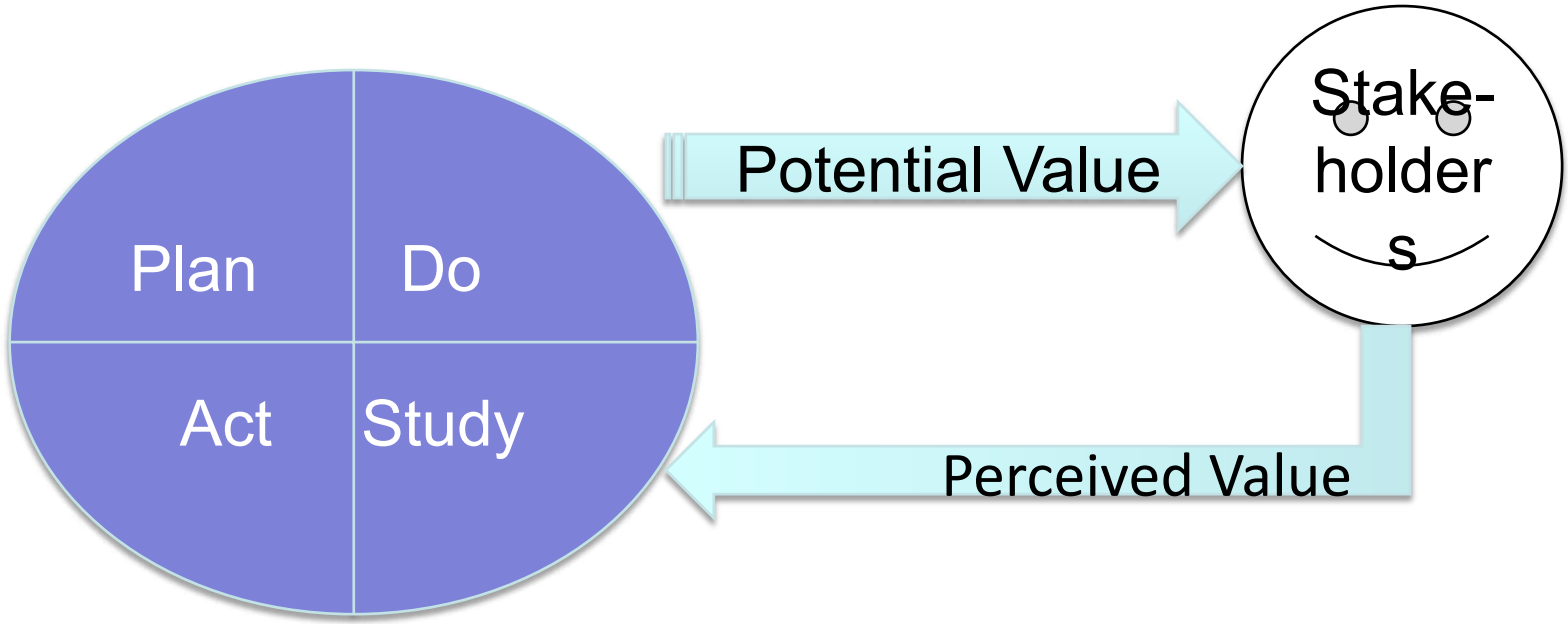
Evo Concept: Potential Value to *Many*



•! Incremental Value Deliveries to ***Many*** Stakeholders

Evo Concept: Short Term Feedback

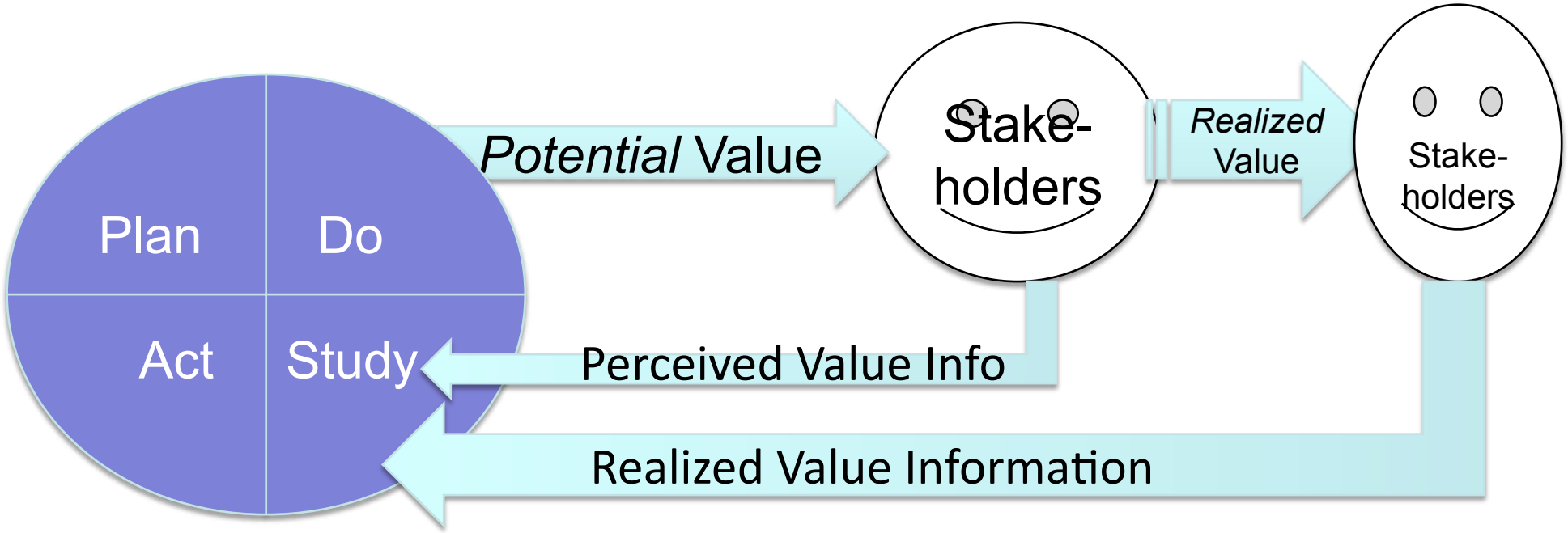
"This looks like a change I can get value from!"



- ! Initial Feedback from Stakeholders, after Evo Cycle delivery

Long-Term *Real* Value Feedback

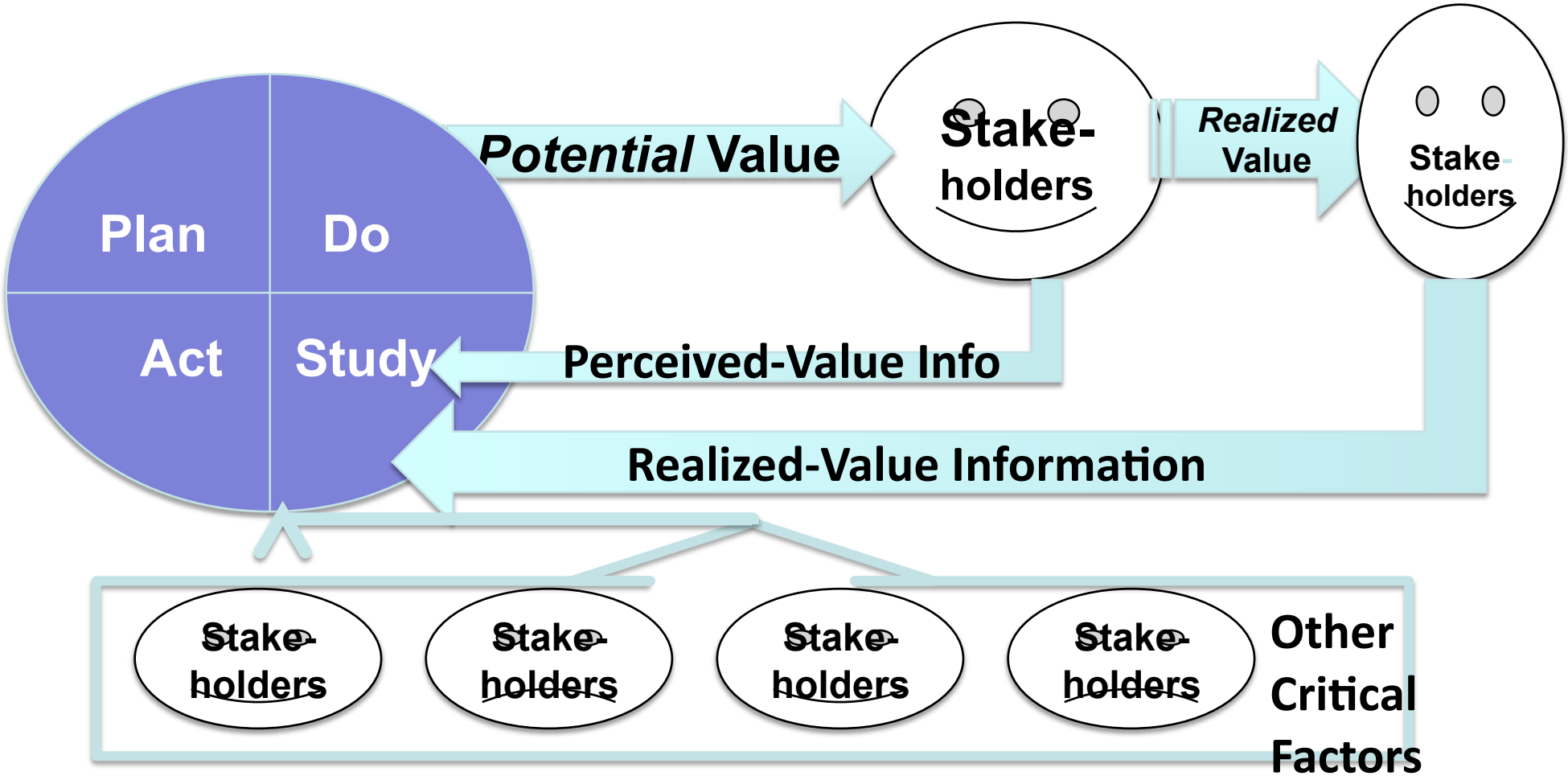
"This is the real value we have gotten to date, and what we expect to get in the future!"



- ! 2 Kinds of Feedback from Stakeholders, when value increment is *really* exploited in practice after delivery

Study critical factors in your environment

"Budget cut, Deadline nearer, New CEO, Cheaper Technology"



- 2 Kinds of Feedback from Stakeholders, when value increment is *really* exploited in practice after delivery.
- Combined with other information from the relevant environment. Like budget, deadline, technology, politics, laws, marketing changes.

Gilb's Evo Method Used Widely at HP and Studied 'Scientifically'

RAPID AND FLEXIBLE PRODUCT DEVELOPMENT: AN ANALYSIS OF SOFTWARE PROJECTS AT HEWLETT PACKARD AND AGILENT

by

Sharma Upadhyayula

M.S., Computer Engineering
University of South Carolina, 1991

Submitted to the System Design and Management Program in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management

at the
Massachusetts Institute of Technology

January 2001

Signature of Author .....

http://www.gilb.com/community/tiki-download_file.php?fileId=65

Sharma Upadhyayula MIT Study Sample Based on Gilb's Evo Projects

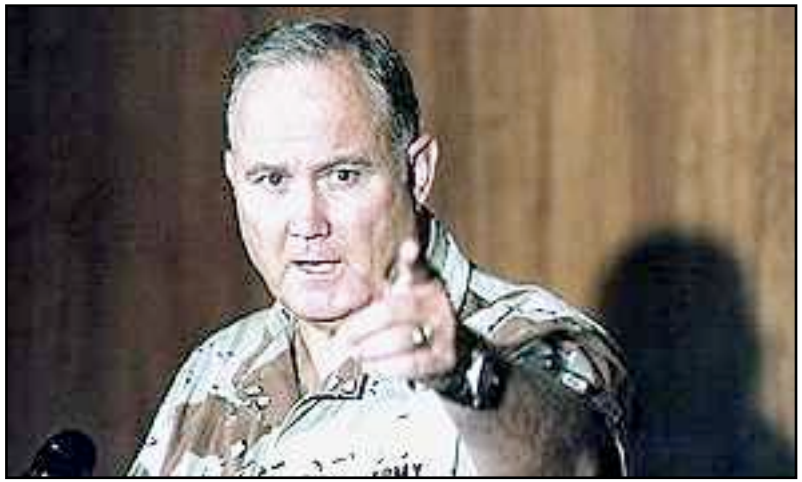
		% Original Features implemented	Bugginess (per mil LOC)	% Schedule Estimation Error	Productivity	Schedule and Budget Perf. perception rating	Customer satisfaction perception rating	% final product functionality in first prototype	% final product functionality in first system integration	% final product functionality in first beta
% Original Features implemented	Correlation Coefficient	1.000	.275	-.250	-.255	.301	-.071	.194	.373	.639**
	Sig. (2-tailed)	.	.255	.288	.277	.197	.767	.425	.116	.003
	N	20	19	20	20	20	20	19	19	19
Bugginess (per mil LOC)	Correlation Coefficient	.275	1.000	-.032	.039	.278	.245	.664**	.198	.357
	Sig. (2-tailed)	.255	.	.898	.875	.249	.311	.003	.432	.146
	N	19	19	19	19	19	19	18	18	18
% Schedule Estimation Error	Correlation Coefficient	-.250	-.032	1.000	.226	-.190	-.060	-.055	.022	-.493*
	Sig. (2-tailed)	.288	.898	.	.287	.423	.802	.809	.923	.017
	N	20	19	24	24	20	20	22	22	23
Productivity	Correlation Coefficient	-.255	.039	.226	1.000	-.496*	-.071	-.202	-.124	-.234
	Sig. (2-tailed)	.277	.875	.287	.	.026	.765	.367	.583	.283
	N	20	19	24	24	20	20	22	22	23
Schedule and Budget Perf. perception rating	Correlation Coefficient	.301	.278	-.190	-.496*	1.000	.072	.247	.112	.464*
	Sig. (2-tailed)	.197	.249	.423	.026	.	.762	.308	.647	.046
	N	20	19	20	20	20	20	19	19	19
Customer satisfaction perception rating	Correlation Coefficient	-.071	.245	-.060	-.071	.072	1.000	.255	-.545*	-.222
	Sig. (2-tailed)	.767	.311	.802	.765	.762	.	.292	.016	.361
	N	20	19	20	20	20	20	19	19	19
% final product functionality in first prototype	Correlation Coefficient	.194	.664**	-.055	-.202	.247	.255	1.000	.518*	.491*
	Sig. (2-tailed)	.425	.003	.809	.367	.308	.292	.	.013	.020
	N	19	18	22	22	19	19	22	22	22
% final product functionality in first system integration	Correlation Coefficient	.373	.198	.022	-.124	.112	-.545*	.518*	1.000	.521*
	Sig. (2-tailed)	.116	.432	.923	.583	.647	.016	.013	.	.013
	N	19	18	22	22	19	19	22	22	22
% final product functionality in first beta	Correlation Coefficient	.639**	.357	-.493*	-.234	.464*	-.222	.491*	.521*	1.000
	Sig. (2-tailed)	.003	.146	.017	.283	.046	.361	.020	.013	.
	N	19	18	23	23	19	19	22	22	23

**. Correlation is significant at the .01 level (2-tailed).

*. Correlation is significant at the .05 level (2-tailed).

Table 3-2 - Market and Technical Feedback Correlation Table – without the outlier in productivity

The Persinscom IT System Case



US Army Example: PERSINSCOM: Personnel System

STRATEGIES ➔ **OBJECTIVES**	Technology Investment	Business Practices	People	Empow-erment	*Principles of IMA Management*	Business Process Re-engineering	SUM
Customer Service ?➔0 Violation of agreement	50%	10%	5%	5%	5%	60%	185%
Availability 90% ➔ 99.5% Up time	50%	5%	5-10%	0	0	200%	265%
Usability 200 ➔ 60 Requests by Users	50%	5-10%	5-10%	50%	0	10%	130%
Responsiveness 70% ➔ ECP's on time	50%	10%	90%	25%	5%	50%	180%
Productivity 3:1 Return on Investment	45%	60%	10%	35%	100%	53%	303%
Morale 72 ➔ 60 per mo. Sick Leave	50%	5%	75%	45%	15%	61%	251%
Data Integrity 88% ➔ 97% Data Error %	42%	10%	25%	5%	70%	25%	177%
Technology Adaptability 75% Adapt Technology	5%	30%	5%	60%	0	60%	160%
Requirement Adaptability ? ➔ 2.6% Adapt to Change	80%	20%	60%	75%	20%	5%	260%
Resource Adaptability 2.1M ➔ ? Resource Change	10%	80%	5%	50%	50%	75%	270%
Cost Reduction FADS ➔ 30% Total Funding	50%	40%	10%	40%	50%	50%	240%
SUM IMPACT FOR EACH SOLUTION	***482%***	***280%***	***305%***	***390%***	***315%***	***649%***	
Money % of total budget	15%	4%	3%	4%	6%	4%	
Time % total work months/year	15%	15%	20%	10%	20%	18%	
SUM RESOURCES	***30***	***19***	***23***	***14***	***26***	***22***	
BENEFIT/RESOURCES RATIO	***16:1***	**14:7**	**13:3**	**27:9**	**12:1**	**29:5**	

Sample of Objectives/Strategy definitions

•! *Example of one of the Objectives:*

Customer Service:

Type: Critical Top level Systems Objective

Gist: Improve customer perception of quality of service provided.

Scale: Violations of Customer Agreement per Month.

Meter: Log of Violations.

Past [Last Year] Unknown Number ←State of PERSCOM Management Review

Record [NARDAC] 0 ? ← NARDAC Reports Last Year

Fail : <must be better than Past, Unknown number> ←CG

Goal [This Year, PERSINCOM] 0 “Go for the Record” ← Group SWAG

Technology Investment:

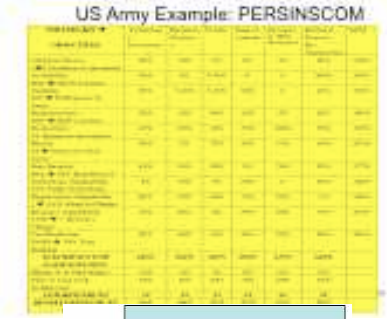
Exploit investment in high return technology.

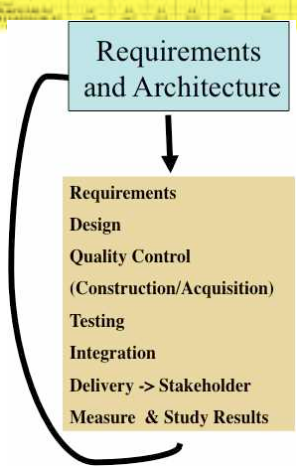
Impacts: productivity, customer service and conserves resources.

•! *An example of one of the strategies defined.*

The Evo Planning Week at DoD

- **Monday**
  - Define top Ten critical objectives, quantitatively
  - Agree that thee are the main points of the effort/project
- **Tuesday**
  - Define roughly the top ten most powerful strategies, for enabling us to reach our Goals on Time
- **Wednesday**
  - Make an Impact Estimation Table for Objectives/Strategies
  - Sanity Test: do we seem to have enough powerful strategies to get to our Goals, with a reasonable safety margin?
- **Thursday**
  - Divide into rough delivery steps (annual, quarterly)
  - Derive a delivery step for 'Next Week'
- **Friday**
  - Present these plans t… …nager (Brigadier General Palicci)
  - get approval to deliv…





Next weeks Evo Step??

- “You won’t believe we never thought of this, Tom!’
- The step:
  - When the Top General Signs in
  - Move him to the head of the queue
    - Of all people inquiring on the system.

The fundamentals of an Evo step: Decomposing for early competitive advantage

- An Evo step must
  - Try to deliver **some** planned function and/ or performance values to **some** stakeholders
  - Maximize the **efficiency** (value to cost ratio) of the delivery
  - Give useful **feedback** before scaling up (risk management)
  - Give project teams practical **experience** in technology, engineering processes, and stakeholder feedback

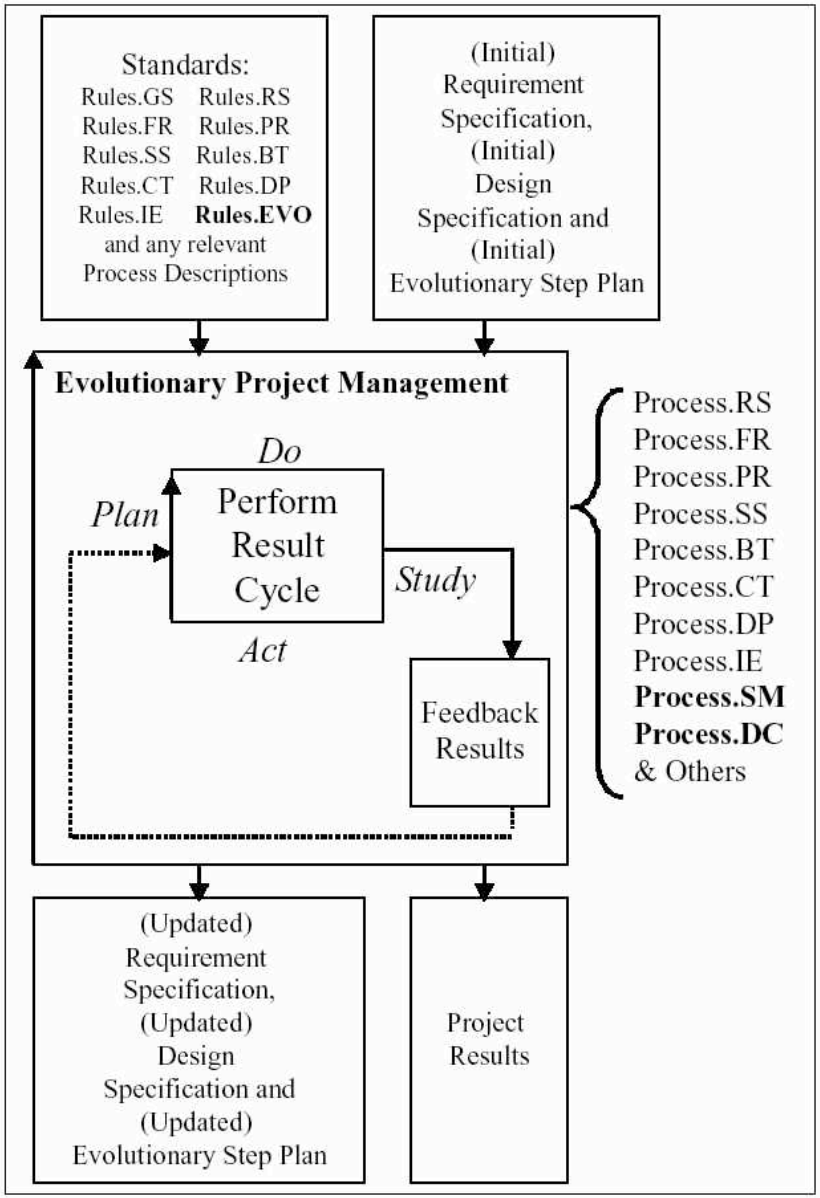
How to decompose systems into small evolutionary steps: (a list of practical tips)

1 Believe there is a way to do it, you just have not found it yet!
I have never seen an exception in 33 years of doing this within many varied cultures.
2 Identify obstacles, but don't use them as excuses: use your imagination to get rid of them!
3 Focus on some usefulness for the stakeholders: users, salesperson, installer, testers or customer. However small the positive contribution, something is better than nothing.
4 Do not focus on the design ideas themselves, they are distracting, especially for small initial cycles. Sometimes you have to ignore them entirely in the short term!
5 Think one stakeholder. Think 'tomorrow' or 'next week.' Think of one interesting improvement.
6 Focus on the results (You should have them defined in your targets. Focus on moving *towards* the Plan levels).
7 Don't be afraid to use temporary-scaffolding designs. Their cost must be seen in the light of the value of making some progress, and getting practical experience.
8 Don't be worried that your design is inelegant; it is results, that count, not style.
9 Don't be afraid that the stakeholders won't like it. If you are focusing on the results they want, then by definition, they should like it. If you are not, then do!
10 Don't get so worried about "what might happen afterwards" that you can make no practical progress.
11 You cannot foresee everything. Don't even think about it!
12 If you focus on helping your stakeholder in practice, now, where they really need it, you will be forgiven a lot of 'sins'!
13 You can understand things much better, by getting some practical experience (and removing some of your fears).
14 Do early cycles, on *willing local mature* parts of your user/stakeholder community.
15 When some cycles, like a purchase-order cycle, take a long time, initiate them early, and do other useful cycles while you wait. This is called 'backroom concurrent engineering'.
16 If something seems to need to wait for 'the big new system', ask if you cannot usefully do it with the 'awful old system', so as to pilot it realistically, and perhaps alleviate some 'pain' in the old system.
17 If something seems too costly to buy, for limited initial use, see if you can negotiate some kind of 'pay as you really use' contract. Most suppliers would like to do this to get your patronage, and to avoid competitors making the same deal.
18 If you can't think of some useful small cycles, then talk directly with the real 'customer', stakeholders, or end user. They probably have dozens of suggestions.
19 Talk with end users and other stakeholders in any case, they have insights you need.
20 Don't be afraid to use the old system and the old 'culture' as a launching platform for the radical new system. There is a lot of merit in this, and many people overlook it.

How does Planguage support Evo project management?

Planguage makes sure we are continuously focused on our clear competitive goals

- Well-defined requirements are the *project management*
  - result delivery targets and
  - constraints
- Well-defined *designs*, and quantified *impact estimates* help *control*
  - the delivery and
  - implementation process



How do you plan an Evo step in Planguage?
By Being explicit about Competitiveness of the Step!

Step Name: Tutorial [7777, Basic].
Stakeholder: Marketing, XX (<agreed, Next Friday>).
Step Implementor: <XX>.
Step Content: HCTD :<Hard Copy Text document> <- Can do 1 week MMM.
. Basic minimal functions
. Step by Step Instructions, in English
. Focus on sales aspects, not how to do it (not yet, in this step)
. Go to specific web sites
. Pinpoint some characteristics of what we see on the terminal
. Compared with what we see on a PC or other terminal
. What instructions should be on the terminal to begin
. Questionnaire for Stakeholder
. Intended audience: Marketing
. Process for Testing with Stakeholder (example observation, times)
. No illustrations, just text.

Step Value: Stakeholder: TTT: Saleability: <some possibility of value>.
Stakeholder: Developers: <value of feedback on a tutorial>.
Step Cost: 10 hours per page, < 10 hours <-MMM.
Step Constraints: Must be deliverable within 1 calendar week.
At Least 3 hours of TTT's time for input and trial feedback.
Step Dependencies: <Feature list of WWW and 7777 WWW Browser> <-MMM.

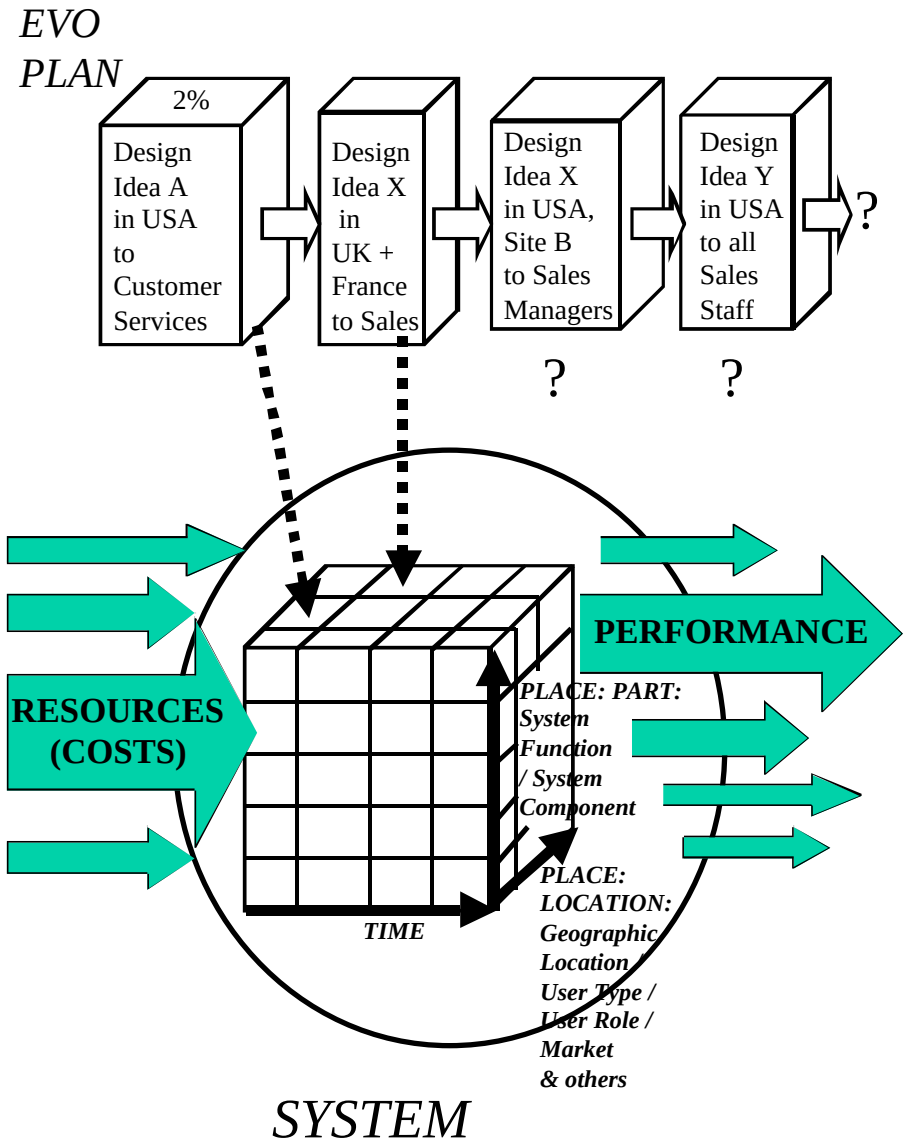
How does Evo relate to requirements?

Step-> Target Require-ment	**STEP1** Plan % (of Target)	actual %	deviation %	**STEP2** to **STEP20** Plan %	plan cumulated to here %	**STEP21** **[CA,NV,WA]** Plan %	plan cumulated to here %	**STEP22** [all others] Plan %	plan cumulated to here %
PERF-1	*5*	*3*	*-2*	*40*	*43*	*40*	*83*	*-20*	*63*
PERF-2	*10*	*12*	*+2*	*50*	*62*	*30*	*92*	*60*	*152*
PERF-3	*20*	*13*	*-7*	*20*	*33*	*20*	*53*	*30*	*83*
COST-A	*1*	*3*	*+2*	*25*	*28*	*10*	*38*	*20*	*58*
COST-B	*4*	*6*	*+2*	*38*	*44*	*0*	*44*	*5*	*49*

- Evo relates directly, measurably, testably, early and frequently to unfulfilled requirements.
- Evo is always seeking the most efficient way to close the requirements gap and complete a project
- The primary measure of Evo project progress is the degree of stakeholder satisfaction (in terms of agreed requirements) as a result of delivered Evo steps.

How does Evo relate to Design?
By Making Sure the Most Competitive Designs are delivered early and provably

- Evo implements designs ***selectively*** depending on priority.
- Designs can be implemented ***partially*** (example in one geographic market or system component) in a *single* step.
- Evo allows us to **be sure** that the designs give *maximum value/cost*
- Evo allows us to **verify**
  - *by measurement*
  - that designs deliver value/cost estimated
  - *before* we commit on a large scale



How does Evo relate to Risk?

It gives excellent practical control over risks to your competitiveness

- Evo reduces risk of deviation from plans
  - By doing projects in **early** and **small** increments
  - By '**learning**' from practical experience
  - And **correcting** bad specifications
  - By grasping and integrating **new opportunities** outside the project (technology, customer, economics)

BASIC EVO PLANNING PolICY!

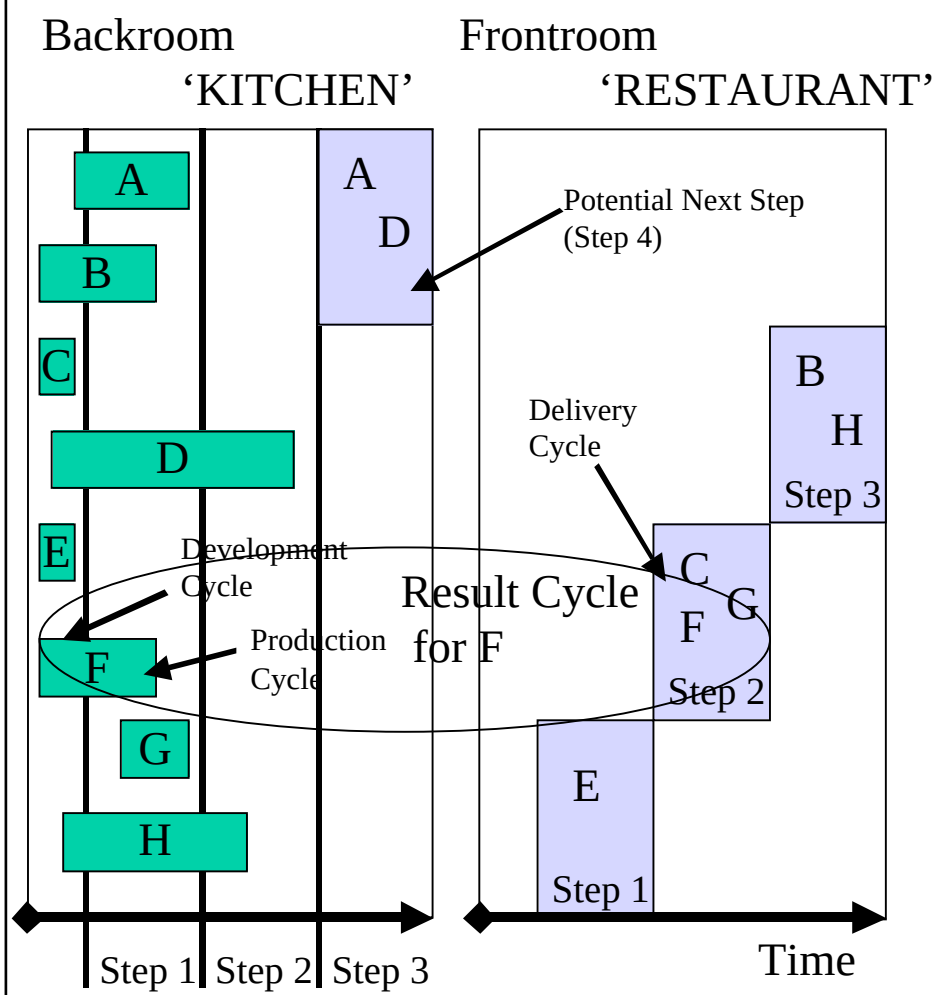
1:**Financial Budget**: No project cycle shall exceed 2% of total financial budget before delivering some measurable, required results to the user.!

2:**Deadline**: No project cycle will exceed 2% of total project time (one week for a one year project) before delivering some measurable, required results to the user.!

3:**Priority**: Project cycles which provide the best ratio of required results to utilized resources (highest benefit-to-cost ratios), must be delivered first to the stakeholders.!

How does Evo relate to process improvement?

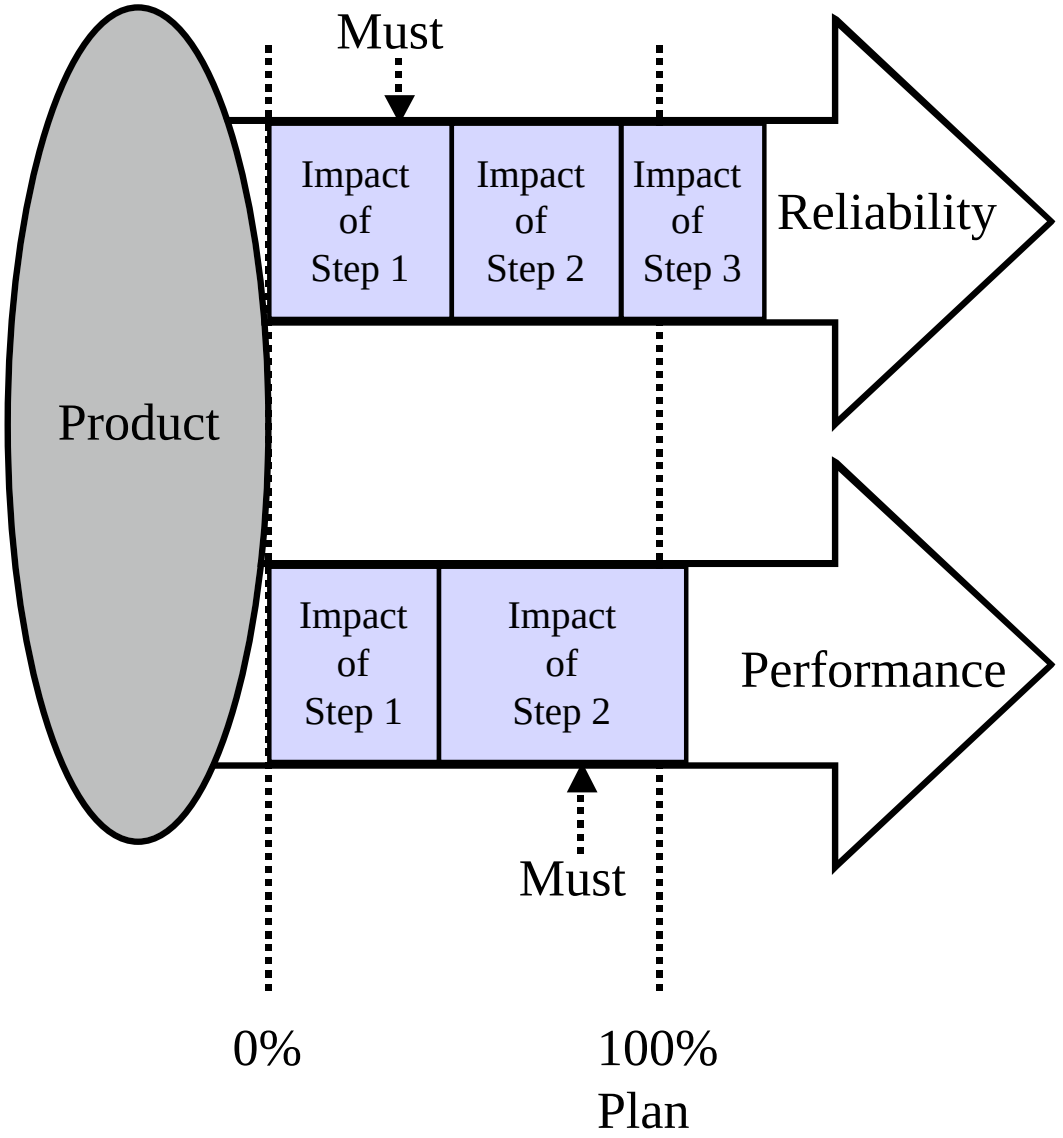
- Evo can measure
  - the success of current processes against expectations,
  - or new experimental ones against expectations
- Evo can signal the need for process improvement and verify that such improvement has taken place
- Evo can help you
  - *early* in the project,
  - continuously,
  - and helps to *train* new people
    - in the adopted processes
    - by frequent cycles of practice and feedback



Degree of Backroom Task Completed during the Frontroom Step Delivery Cycle

How does Evo relate to competitiveness?

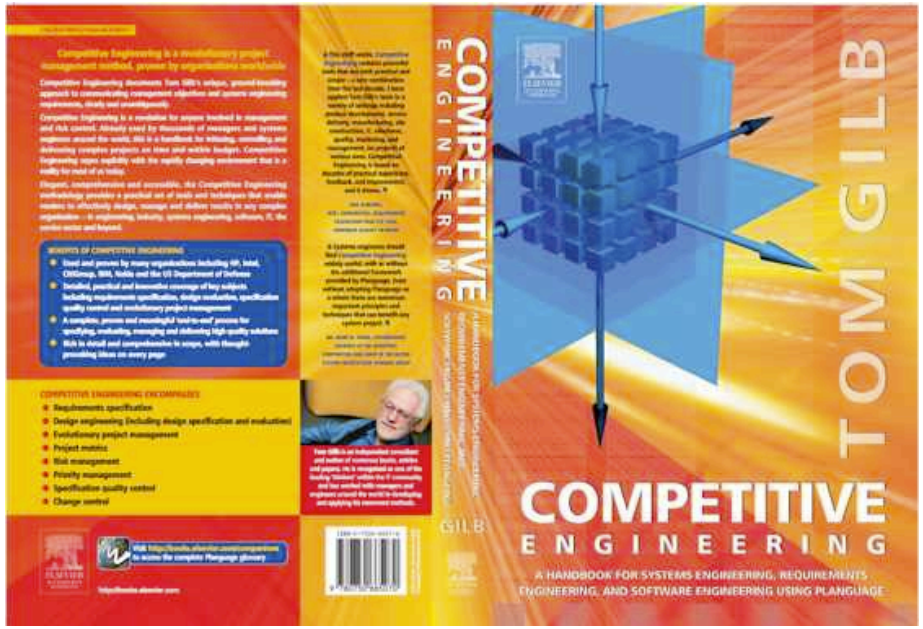
- Evo is focused on delivery of quantified specified stakeholder value
- Evo is 'agile'
  - and can change plans, designs, processes, and requirements -
  - in order to deliver the most competitive solutions
  - early, gradually, and with smart priorities.

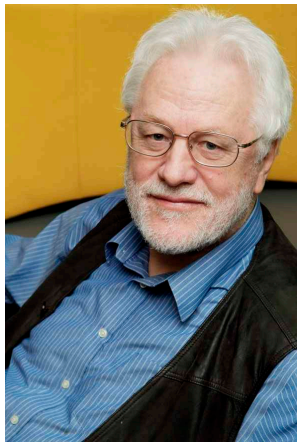


Summary

Planguage gives you ***tools*** to be more competitive.

- The entire set of Planguage tools also applies to
  - software engineering
  - and top management planning
    - (see 'Priority Management' book at www.gilb.com)



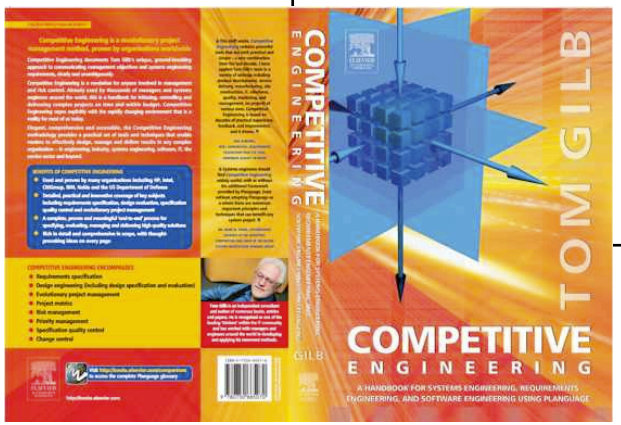


If we have more time ….

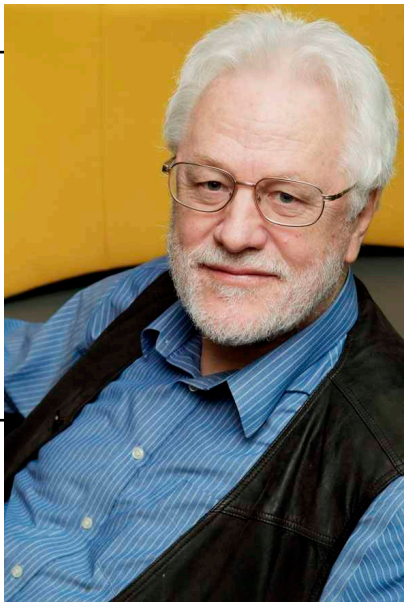
- ! Or we might skip to these during the main presentation

Designing **Maintainability** in Software Engineering: a *Quantified* Approach.

Tom Gilb
Result Planning Limited
Tom.Gilb@INCOSE.org

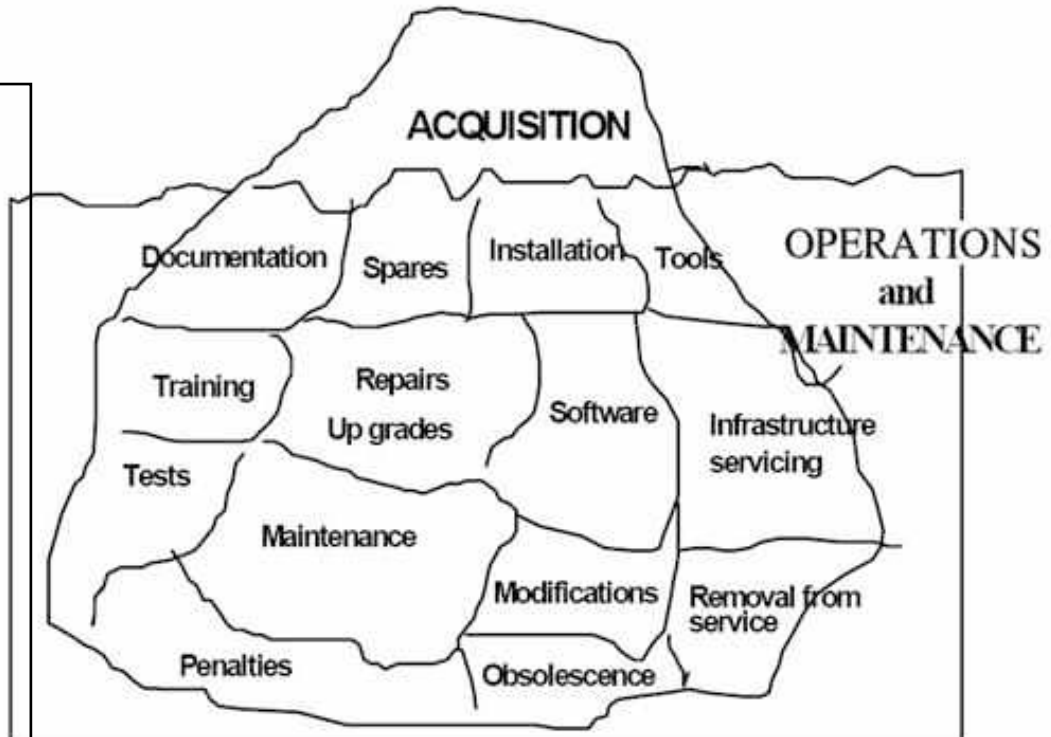


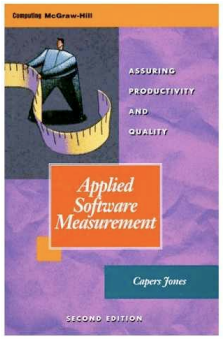
Version April 15 2008



Abstract.

- ! Software system maintenance costs are a substantial part of the life cycle costs.
- ! They can easily steal all available effort away from new development.





System Lifetime Expectancy: Capers Jones

S

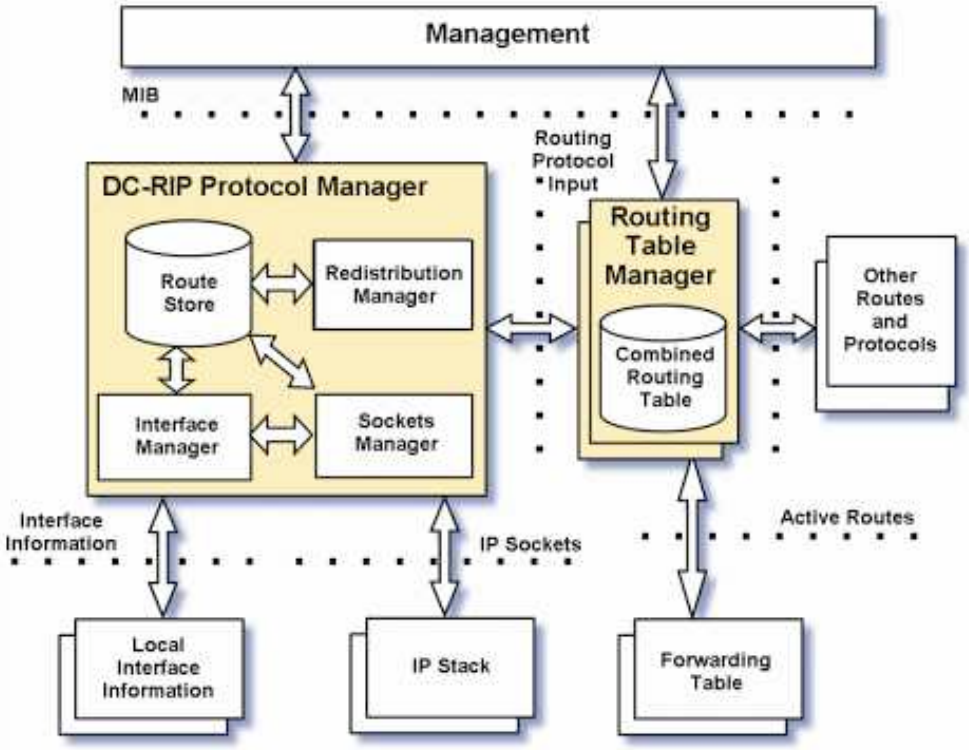
Table 30: Estimated Life Expectancy of Applications before Retirement or Replacement

(Note: Data is expressed in terms of calendar years from first deployment until last retirement. Length of service is proportional to size.)

	MIS Projects	Web Projects	Domestic Outsource Projects	Systems & Embedded Projects	Commercial Projects	Civilian Government Projects	Military Projects	*Average*
Size in FP								
1	1.40	1.00	1.50	3.00	2.00	2.00	3.00	*1.99*
10	2.50	2.00	3.00	4.00	3.00	4.00	4.00	*3.21*
100	4.00	3.00	4.50	4.50	4.00	5.50	5.00	*4.36*
1,000	5.00	4.00	5.00	6.00	5.00	8.00	9.00	*6.00*
10,000	18.00	9.00	14.00	13.00	9.00	22.00	23.00	*15.43*
100,000	20.00	10.00	17.00	15.00	14.00	24.00	24.00	*17.71*
1,000,000	25.00	12.00	27.00	18.00	20.00	28.00	26.00	*22.29*
Average	*10.84*	*5.86*	*10.29*	*9.07*	*8.14*	*13.36*	*13.43*	*10.14*

Abstract

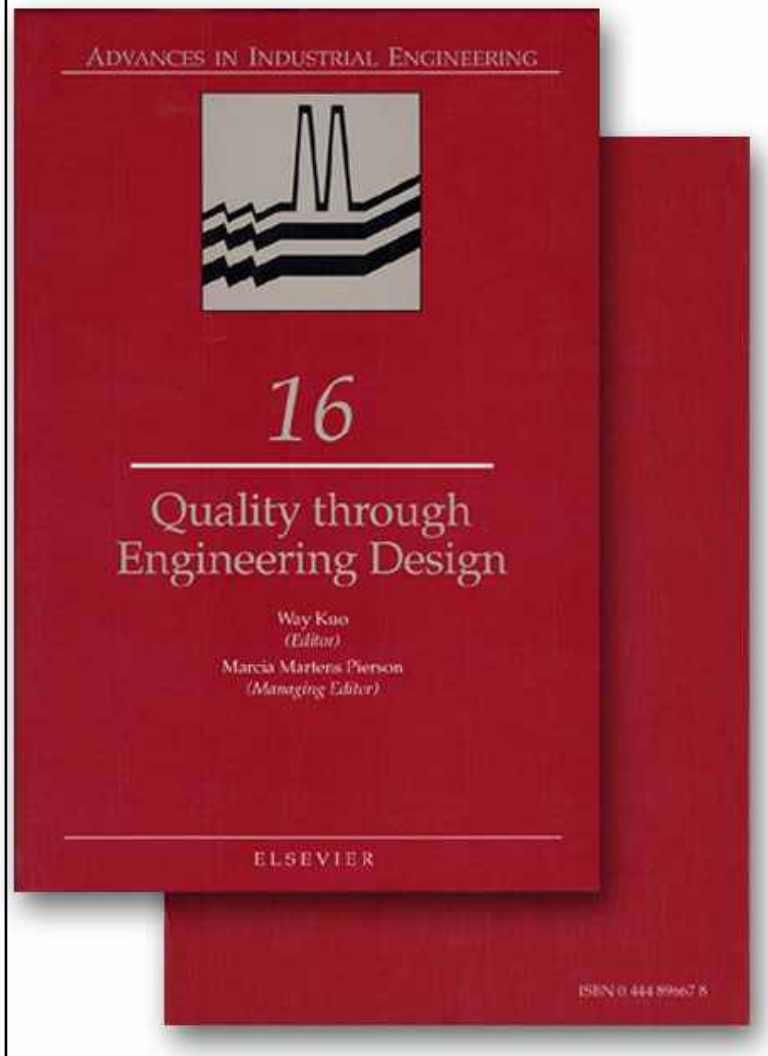
- I believe that this is because
  - **maintainability is, as good as never, systematically engineered into the software.**
- Our so-called software architects bear a primary responsibility for this, but they do not engineer to targets.
- They just throw in customs and habits that seem appropriate.



Did you ever see ideas like performance and quality, for example 'Portability Levels' ! in a software architecture diagram?!

Abstract

- We need to
  - define our maintainability requirements quantitatively,
  - Set quality investment targets that will pay off,
  - pursue long-term engineered improvement of the systems, and then
  - ‘architect’ and ‘engineer’ the resulting system.
- Traditional disciplines may already in principle understand this discipline,
  - some may not understand it,
  - some may simply not apply the engineering understanding that is out there



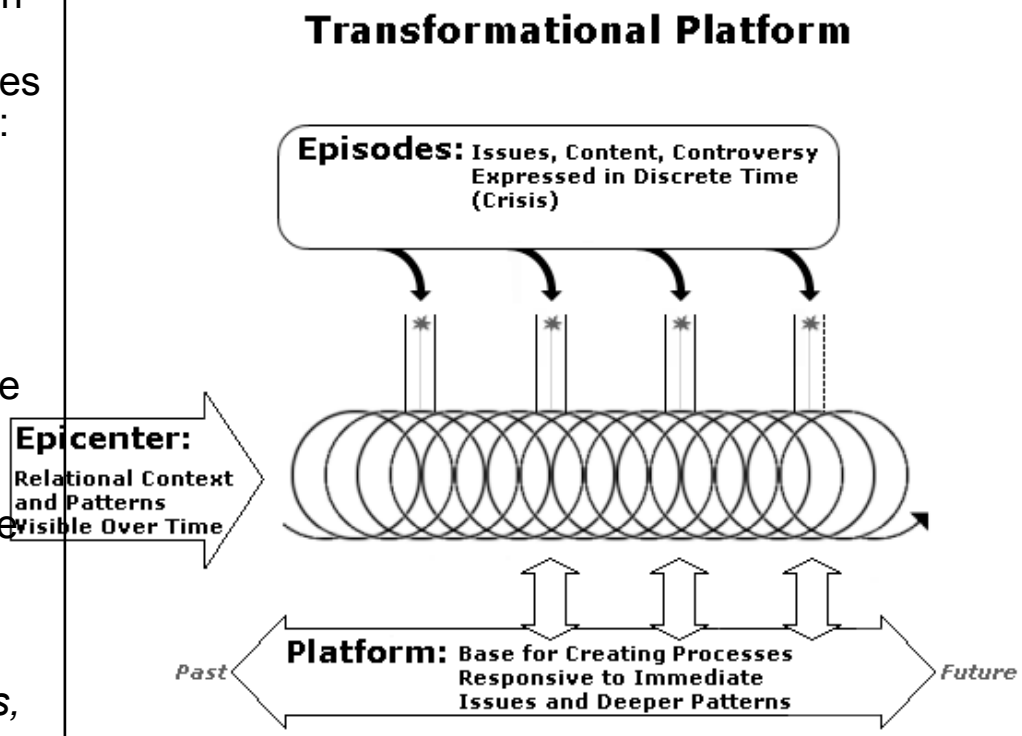
The Maintainability Problem

- Software systems are built under high pressure to meet deadlines, and with initial emphasis on performance, reliability, and usability.
- The software attributes relating to later changes in the software – maintainability attributes are:
  - never specified quantitatively up front in the software quality requirements
  - never architected to meet the non-specified maintainability quality requirements
  - never built to the unspecified architecture to meet the unspecified requirements
  - never tested before software release
  - never measured during the lifetime of the system.

*"A number of people expressed the opinion that code is often **not designed for change**. Thus, while the code meets its operational specification,*

for maintenance purposes it is poorly designed and documented " [Dart 93]

- In short, there is no engineering approach to software maintainability.



What *do* we do in practice today?

- we might *bullet point* some high-level objectives
  - (‘• Easy to maintain’)
  - which are never taken seriously
- we might even decide the technology we will use to reach the vague ideal
  - (“• Easy to maintain through modularization, object orientation and state of the art standard tools”)
- larger institutions might have ‘software architects’ who carry out certain customs, such as
  - decomposition of the software,
  - choice of software platforms and software tools – generally intended to help – hopefully.
  - But with no specific resulting level or type of maintainability in mind.
- we might recommend more and better tools, but totally fail to suggest an *engineering* approach [Dart 93].
- We could call this a ‘craft’ approach.
- It is not ‘engineering’ or ‘architecture’ in the normal sense.

Principles of Systems Maintainability

- I would like to suggest a set of principles about software maintainability,
  - in order to give us a framework:

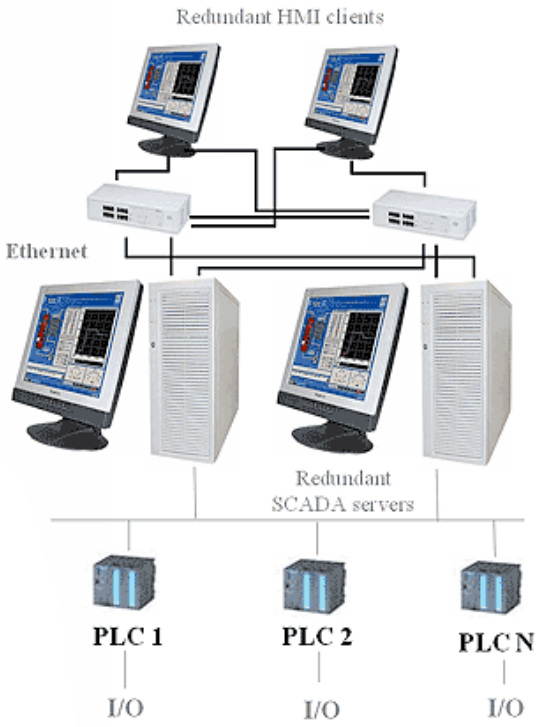
Body Maintenance: {Relax, Exercise, Breathing, Diet, Positive Thinking and Meditation}. !

1. **The Conscious Design Principle**:

- ! Maintainability must be *consciously* designed into a system:
  - ! failure to **design** to a set of levels of maintainability
  - ! means the **resulting maintainability** is both *bad* and *random*.

Conscious Design

- Clarify
  - Robust →
    - 200 Days Between Restarts
- Find Solutions
  - Triple Redundant Systems ?
- Verify Solutions
  - 400 Days average achieved!



2.! The Many-Splendored Thing Principle.

- ! Maintainability is
  - ! a **wide set** of change-quality types,
  - ! under a **wide** variety of **circumstances**:
  - ! so we must clearly define **what quality type** we are trying to engineer. Like:
    - ! Portability, scalability, maintainability?

http://www.youtube.com/watch?v=X-JiKA1vTRo = Nat King Cole "Love is…"!

Real Example of ***Lack*** of Scales (Repeated)

- Notice in this real case
  - No numbers
    - No targets
    - No Constraints
  - No benchmarks
  - No [Qualifiers]
    - Where
    - If
    - Dates
  - No sources
  - No Justifications

*1. Central to The Corporations business strategy is to be the world's **premier** integrated_<domain> service **provider**.*

*2. Will provide a much more efficient **user** experience*

*3. Dramatically scale back the **time** frequently needed after the last data is acquired to time align, depth correct, splice, merge, recompute and/or do whatever else is needed to **generate** the desired **products***

*4. Make the system much **easier** to **understand** and **use** than has been the case for previous system.*

*5. A primary goal is to provide a much more **productive** system **development** environment than was previously the case.*

*6. Will provide a richer set of functionality for **supporting** next-generation logging **tools** and applications.*

*7. **Robustness** is an essential system requirement (see rewrite in example below)*

*8. Major improvements in **data quality** over current practices*

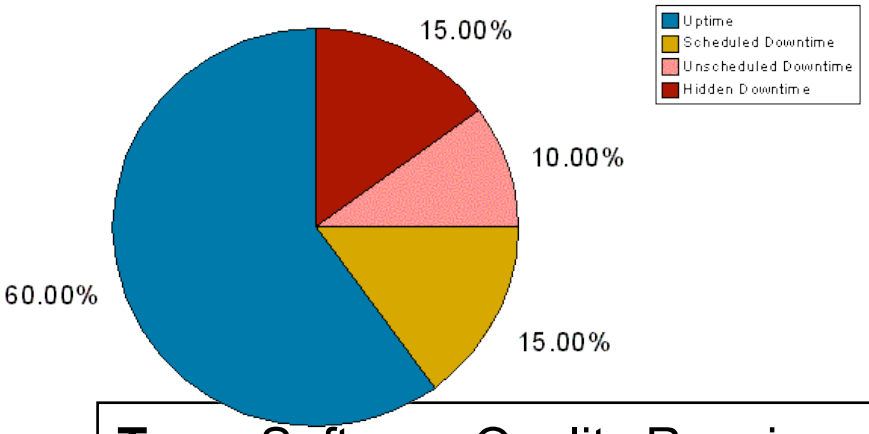
This lack of clarity cost $100,000, 000!

Rock Solid Robustness: *many splendored*

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

-

Software Downtime:



Type: Software Quality Requirement. **Version**: 25 October 2007.
Part of: Rock Solid Robustness.
Ambition: to have minimal downtime due to software failures <- HFA 6.1
Issue: does this not imply that there is a system wide downtime requirement?

Scale: <mean time between forced restarts for defined [Activity], for a defined [Intensity].>

Fail [Any Release or Evo Step, Activity = Recompute, Intensity = Peak Level] 14 days <- HFA 6.1.1

Goal [By 2008?, Activity = Data Acquisition, Intensity = Lowest level] : 300 days ??
Stretch: 600 days.

Restore Speed:

Type: Software Quality Requirement. **Version**: 25 October 2007.
Part of: Rock Solid Robustness
Ambition: Should an error occur (or the user otherwise desire to do so), the system shall be able to restore the system to a previously saved state in less than 10 minutes. <-6.1.2 HFA.

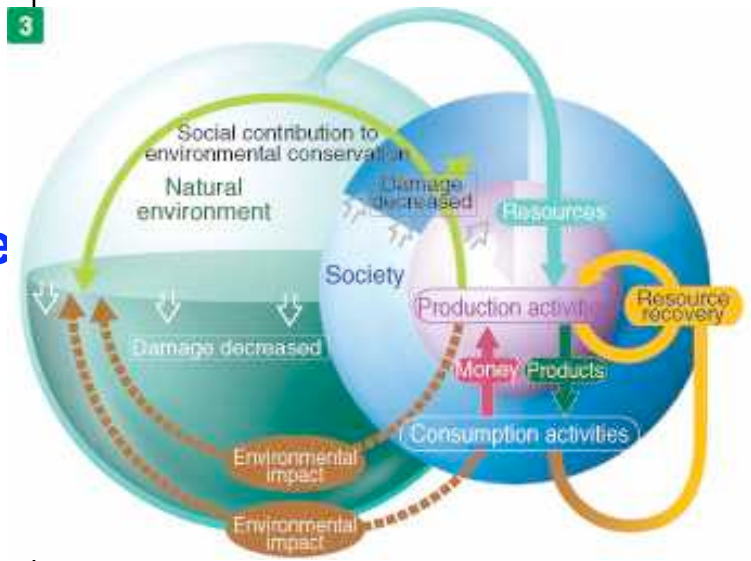
Scale: Duration from Initiation of Restore to Complete and verified state of a defined [Previous: Default = Immediately Previous]] saved state.

Initiation: defined as {Operator Initiation, System Initiation, ?}. Default = Any.

Goal [Initial and all subsequent released and Evo steps] 1 minute?

Fail [Initial and all subsequent released and Evo steps] 10 minutes. <- 6.1.2 HFA

Catastrophe: 100 minutes.



Testability:

Type: Software Quality Requirement.
Part of: Rock Solid Robustness
Initial Version: 20 Oct 2006
Version: 25 October 2007.
Status: Demo draft,
Stakeholder: {Operator, Tester}.
Ambition: Rapid-duration automatic testing of <critical complex tests>, with extreme operator setup and initiation.

Scale: the duration of a defined [Volume] of testing, or a defined [Type], by a defined [Skill Level] of system operator, under defined [Operating Conditions].

Goal [All Customer Use, Volume = 1,000,000 data items, Type = WireXXXX Vs DXX, Skill = First Time Novice, Operating Conditions = Field, {Sea Or Desert}. <10 mins.

Design Hypothesis: *Tool Simulators, Reverse Cracking Tool, Generation of simulated telemetry frames entirely in software, Application specific sophistication, for drilling – recorded mode simulation by playing back the dump file, Application test harness console <-6.2.1 HFA*

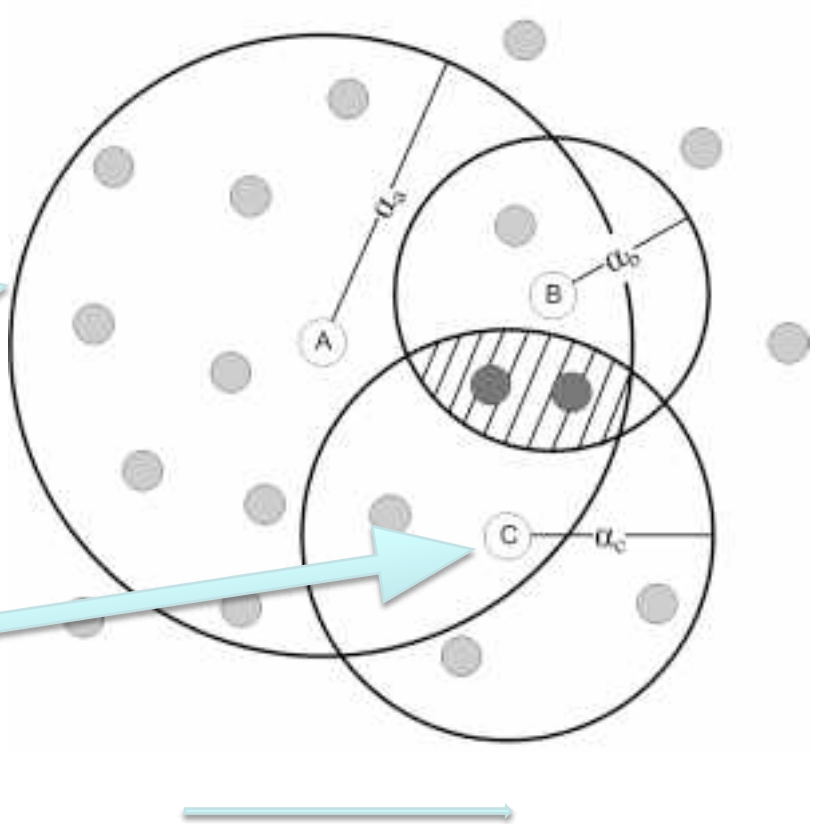
Another Real (Doctored) Example: Financial Corp. Top Level Project requirements

DO YOU SEE ANYTHING RELATED TO MAINTAINABILITY?

1. *Reduce the **costs** associated with managing redundant / regionally **disparate** systems.*
2. ***Single** global portfolio management system.*
3. *Reduce overall **spending** with a reduction in redundant initiatives.*
4. *Governance structures - system agnostic.*
5. *All projects in project portfolio system.*
6. ***Reduce development** project **spend** on low priority work with better alignment between Technology and business demand.*
7. *Project portfolio Framework, Business Value metrics for **prioritization**.*
8. ***Reduction** in **cost** over runs.*
9. ***Definition** criteria for project **success**.*
10. *Metrics and exception reporting for **cost** management.*
11. *Linkage of actual **costs** to forecast.*
12. *Increase **revenue** with a faster **time to market**.*
13. *Knowledge management, project ramp up templates.*

3. **The Multi-Level Requirement Principle.**

- The levels of maintainability we decide to require cab be
  - partly '**constraints**',
    - a necessary minimum of ability to avoid failure,
  - and partly desirable **'target'** levels
    - that are determined by what pays off to invest in.



Software Downtime: Multiple Levels

Type: Software Quality Requirement. **Version**: 25 October 2007.
Part of: Rock Solid Robustness.
Ambition: to have minimal downtime due to software failures <- HFA 6.1
Issue: does this not imply that there is a system wide downtime requirement?

Scale: <mean time between forced restarts for defined [Activity], for a defined [Intensity].>

Fail [Any Release or Evo Step, Activity = Recompute, Intensity = Peak Level] 14 days <- HFA 6.1.1

Goal [By 2008?, Activity = Data Acquisition, Intensity = Lowest level] : 300 days ??

Stretch: 600 days.

Restore Speed: Multiple Levels

***Type**: Software Quality Requirement.* ***Version**: 25 October 2007.*

***Part of**: Rock Solid Robustness*

***Ambition**: Should an error occur (or the user otherwise desire to do so), the system shall be able to restore the system to a previously saved state in less than 10 minutes. <-6.1.2 HFA.*

Scale: Duration from Initiation of Restore to Complete and verified state of a defined [Previous: Default = Immediately Previous]] saved state.

Initiation: defined as {Operator Initiation, System Initiation, ?}. Default = Any.

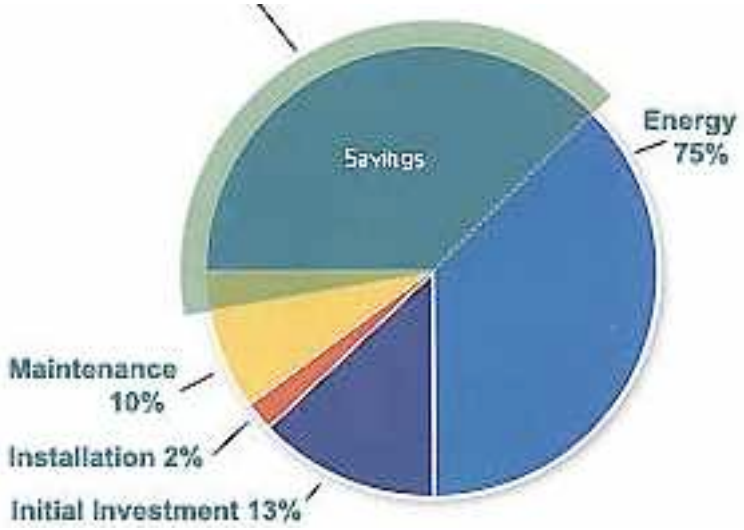
Goal [Initial and all subsequent released and Evo steps] ▮ minute?

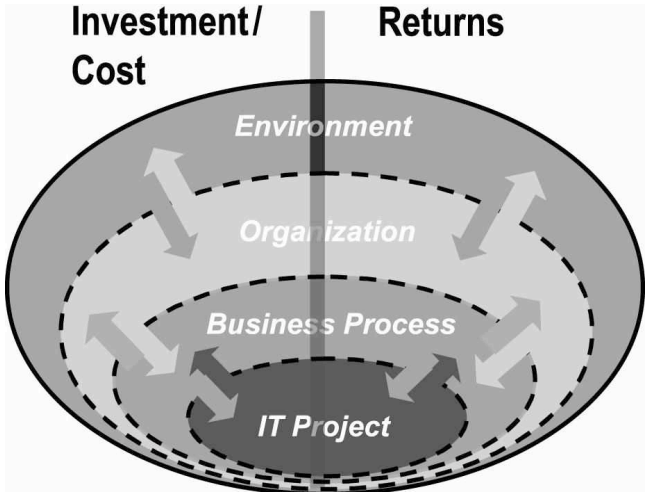
Fail [Initial and all subsequent released and Evo steps] ▮◐ minutes. <- 6.1.2 HFA

Catastrophe: ▮◐◐ minutes.

4. **The Payoff Level Principle.**

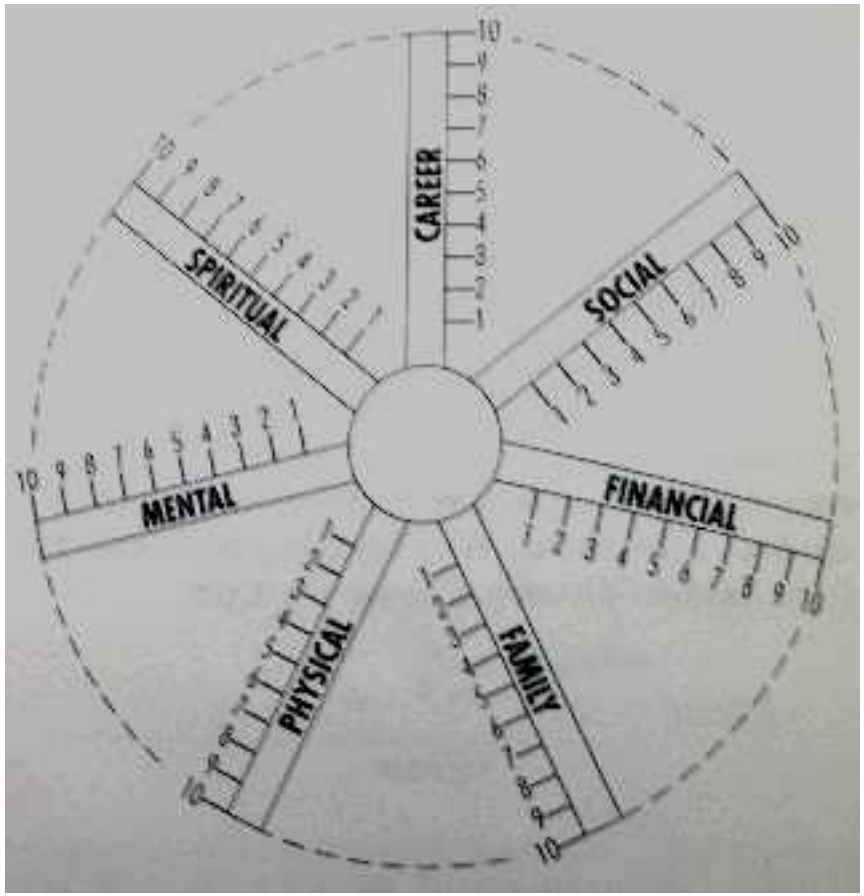
- The *levels of maintainability* it **pays off** to invest in,
  - depend on **many** factors –
  - but certainly on the system **lifetime** expectancy,
  - the **criticality**/illegality/cost of not being able to change correctly or change in time,
  - and the cost and availability of necessary skilled **professionals** to carry out the changes.





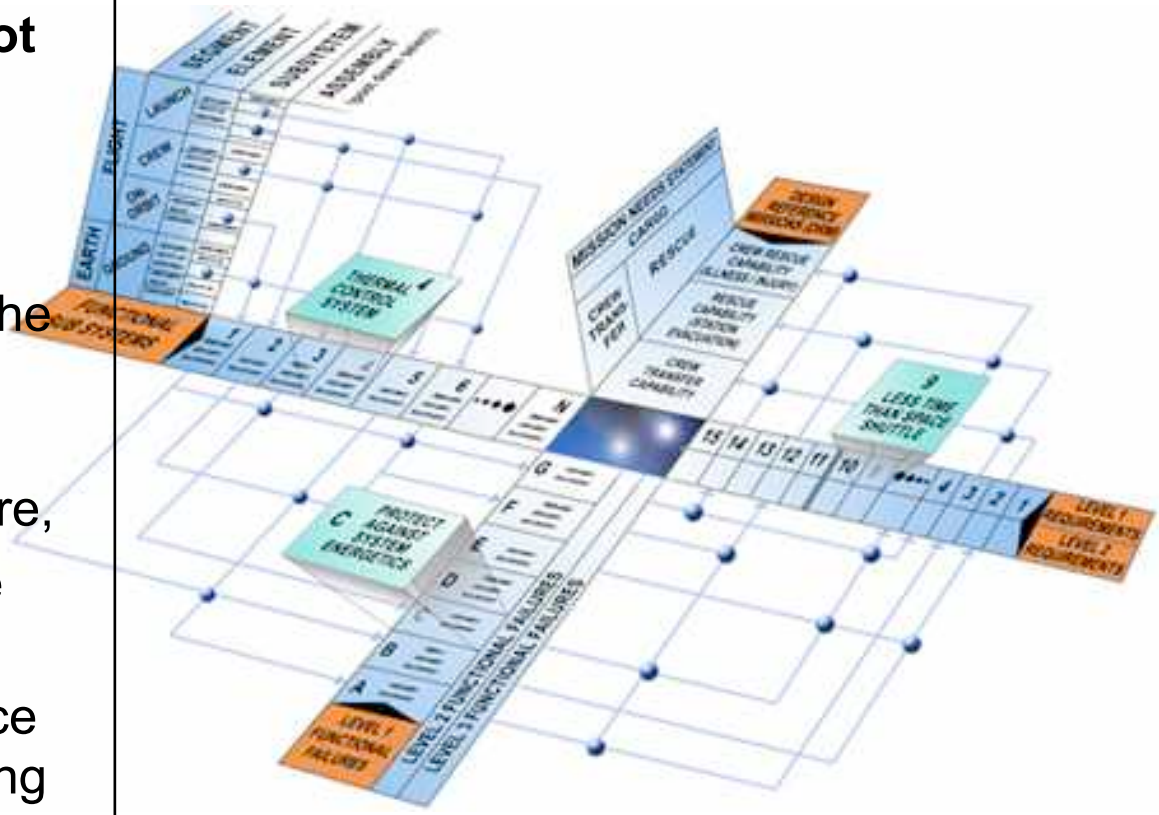
5. **The Priority Dynamics Principle.**

- ! The **maintainability** requirements must *compete for priority*
  - ! for **limited** resources
  - ! with all **other** requirements.
- ! We **cannot** simply **demand** arbitrary *desired* levels of maintainability.



The Engineering Solution

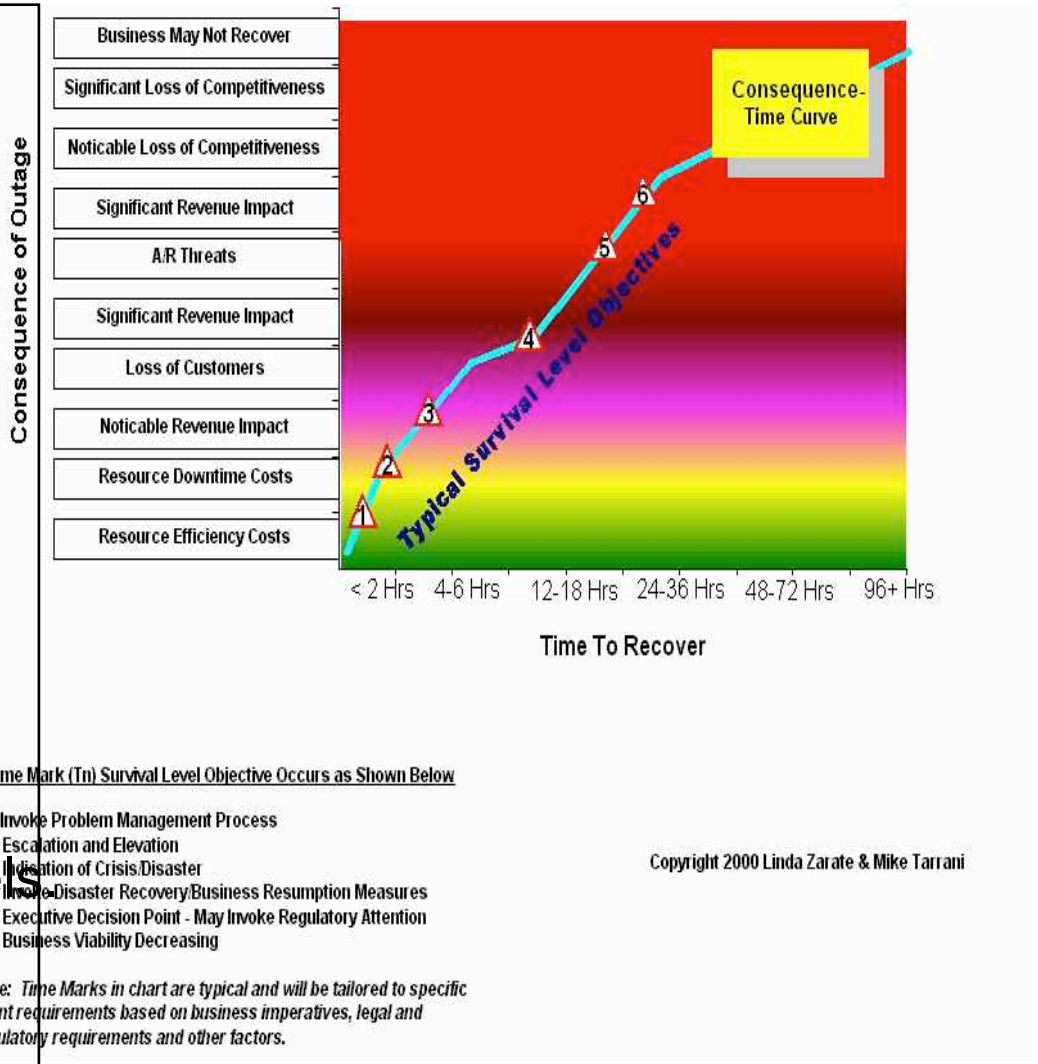
- There are many small and less critical software systems where
  - engineering the maintainability would **not** be interesting,
  - or would **not** pay off.
  - **Nobody** cares.
- This **talk is addressed** to the vast number of current situations where
  - the total **size** of software,
  - the **growth** of software annually,
  - the **cost** of maintenance annually – are all causing management to wonder – '
    - **Is there a better way?'**
-



The method is straightforward, and it is well-understood engineering in 'real' engineering disciplines.

•! In simple terms it is:

1. Define the maintainability requirements *quantitatively*.
2. Design to meet those requirements, if possible and economic.
3. Implement the designs and test that they meet the required levels.
4. Quality Control that the design continues to meet the required maintainability quality levels, and take action in the case of degradation, to get back to current required levels.



Let us take a simplified tour of the method.

Requirement specification (using 'Planguage' [Gilb 2005]:

Bug Fixing Speed:

Type: Software Product Quality Requirement.

Scope: Product Confirmit [Version 12.0 and on]

Ambition Level: Fast enough bug fixing so that it is a non-issue with our customers.

Scale of Measure: **Average Continuous Hours from Bug occurs and is observed in any user environment, until it is correctly corrected and sufficiently tested for safe release to the field, and the change is in fact installed at, at least, one real customer, and all consequences of the bug have been recovered from at the customer level.**

Meter: QA statistics on bug reports and bug fixes.

Past [Release 10.0] 36 hours <- QA Statistics

Fail [Release 12.0, Bug Level = Major] 6 hours <- QA Directors Plan

Goal [Release 12.0, Bug Level = Catastrophic] 2 hours <- QA Directors Plan.

Goal [Release 14.0, Bug Level = Catastrophic] 1 hour <- QA Directors Plan.

→ Next slide!

Planguage Intelligibility

- It should be possible to read this specification,
  - slowly,
  - even for those not trained in Planguage,
  - and to be able to explain exactly what the requirement is.
-
- Notice especially the 'Scale of Measure'.
  - **Scale** of Measure: **Average Continuous Hours from Bug occurs and is observed in any user environment, until it is correctly corrected and sufficiently tested for safe release to the field, and the change is in fact installed at, at least, one real customer, and all consequences of the bug have been recovered from at the customer level.**
  - It encompasses the entire maintenance life cycle
    - from first bug effect observation
    - until customer level correction in practice.
  - *That is a great deal more than just some **programmer staring at code** and **seeing the bug** and **patching it**.*
  - The corresponding **design**
    - will have to encompass **many** processes and technologies.
-

The Breakdown into Sub-problems

Here is a list of the areas we need to design for, and quite possibly have a secondary target level for each:

1. Problem Recognition Time.

How can we reduce the time from bug actually occurs until it is recognized and reported?

2. Administrative Delay Time:

How can we reduce the time from bug reported, until someone begins action on it?

3. Tool Collection Time.

How can we reduce the time delay to collect correct, complete and updated information to analyze the bug: source code, changes, database access, reports, similar reports, test cases, test outputs.

4. Problem Analysis Time.

Etc. for all the following phases defined, and implied, in the Scale scope above.

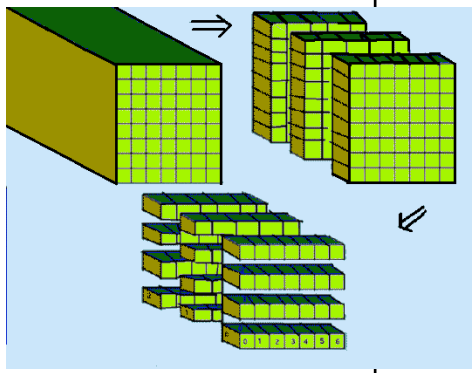
5. Correction Hypothesis Time

6. Quality Control Time

7. Change Time

8. Local Test Time

9. Field Pilot Test Time



10. Change Distribution Time

11. Customer Installation Time

12. Customer Damage Analysis Time

13. Customer Level Recovery Time

14. Customer QC of Recovery Time

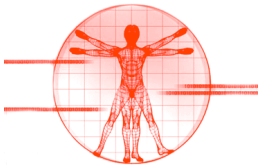
This model is based on one in Ireson (ed.): Reliability Handbook!

Let us take a look at a possible first draft of some design ideas:

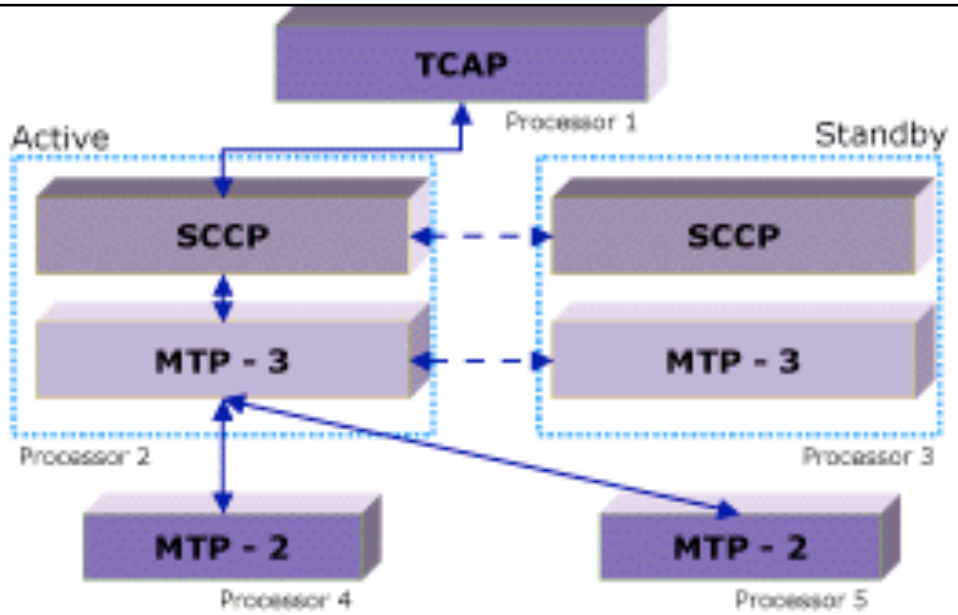
- Note: I have intentionally suggested some *dramatic* architecture,
  - in an effort to meet the *radically* improved requirement level.
- The reader need not take any design *too* seriously.
- This is an example of trying to solve the problem, using engineering techniques (redundancy)
  - that have a solid scientific history.

University of Alaska's ! Museum of the North! in Fairbanks!

1. Problem Recognition Time.

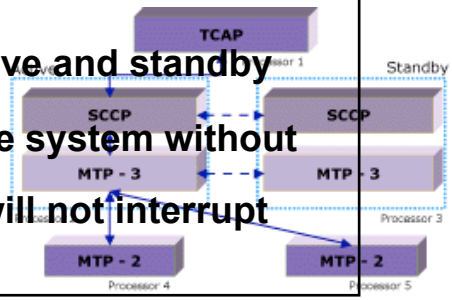


- **Design: Automated N-version distinct software comparison [Inacio 1998]**
  - **at selected critical customer sites,**
  - **to detect potential bugs automatically.**

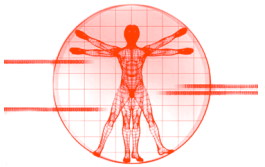


Trillium | Distributed Fault-Tolerant/High-Availability (DFT/HA) Core

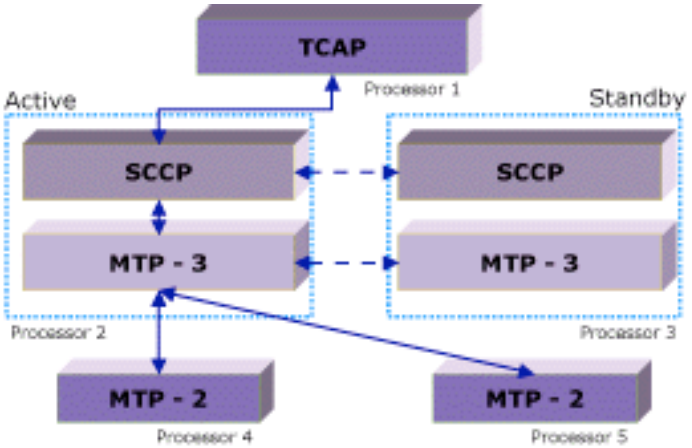
- Complete recovery during failure.
  - This feature is available in both pure fault-tolerant and distributed fault-tolerant systems.
  - When a failure occurs, failed protocol layers are able to completely recover stable state information.
  - All protocol resources present in a stable state during the failure are maintained on the standby.
- Application restart on processor loss.
  - This feature is applicable to pure distributed systems. If a processor in a pure distributed system fails, applications on the failed processor may be restarted on available processors to provide service for subsequent user traffic.
- Survive up to n-1 faults.
  - DFT protocol layers may survive up to n-1 faults without loss of service where n is the number of processors over which the protocol layer was distributed.
  - With the lost application restart feature enabled, a distributed protocol layer may continue to provide full service until the last processor in the system fails.
  - User defined system operations. Advanced distributed system operations such as dynamic load balancing may be implemented using basic services provided by the core software.
- Graceful node shutdown.
  - The system manager provides an operation to gracefully shutdown a node and an option to redistribute the protocol load onto remaining processors in the system
  - . The load redistribution is completely transparent to the system users.
- Maintenance operations.
  - The system manager provides an operation to swap the states of an active and standby node.
  - This functionality may be used to perform maintenance operations on the system without shutting it down
  - . These operations are completely transparent to the system users and will not interrupt service provided by the system.



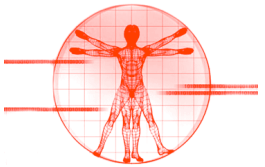
2. Administrative Delay Time:



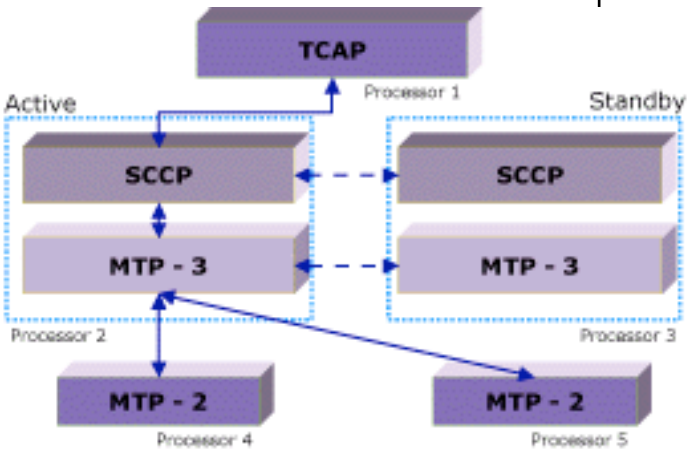
- **Design: Direct digital report**
  - **from distinct software discrepancies**
  - **to our global,**
    - **3 zone,**
    - **24/7**
    - **bug analysis service.**



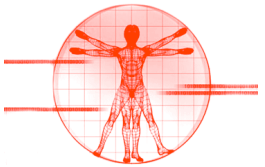
3. Tool Collection Time.



- **Design: All necessary tools are electronic,**
  - **and collection is based on**
    - **customers installed version and its fixes.**
  - **The distinct software, bug capture**
    - **collects local input sequences.**



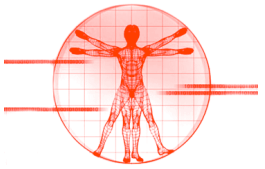
4. Problem Analysis Time.



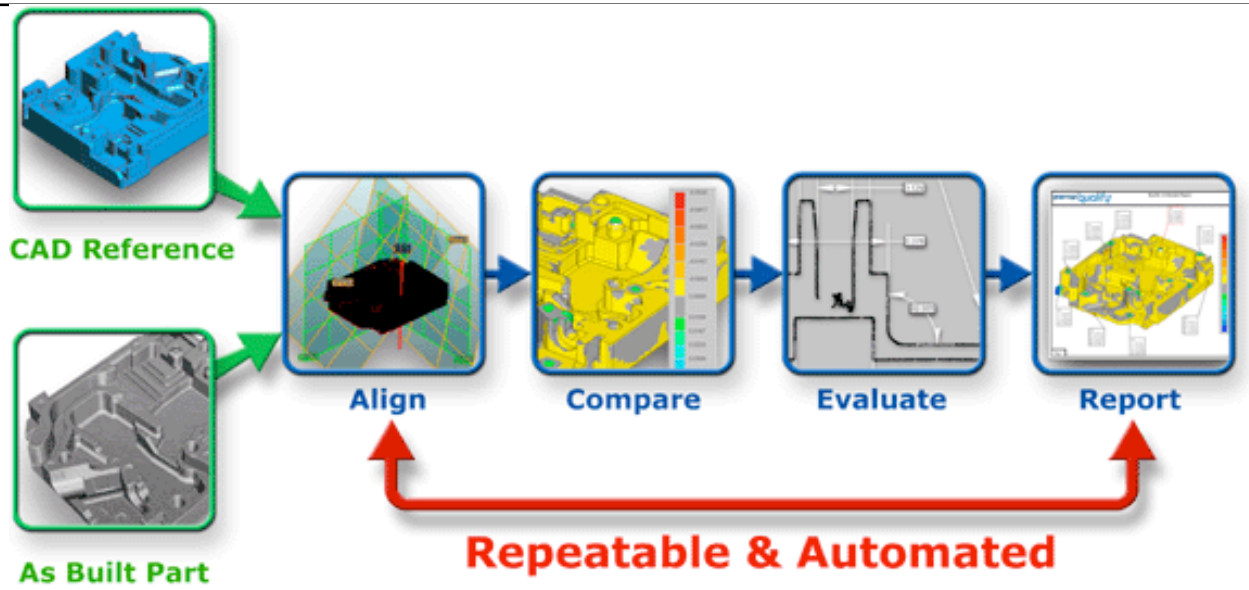
- **Analyst Selection:**
  - **Design: The fastest bug analysts are**
    - **selected based on actual past performance statistics, and**
    - **rewarded in direct relation to their timing**
      - **for analyzing root cause, or correct fix.**

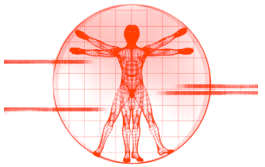
5. Correction Hypothesis Time

- **Design: Same design as Analyst Selection,**
  - **but applies to correct change specification speed statistics.**



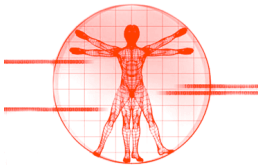
- **Design: Rigorous**
  - **30 minute or less inspection**
  - **of change spec by other bug analysts,**
  - **with reward for finding major defects**
    - **as judged by our defect standards.**





- **Design: Changes are applied**
  - **in parallel with QC,**
  - **and modified only if change defects found in QC.**

8. Local Test Time

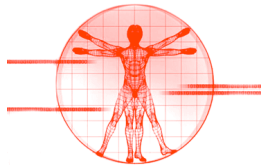


- **Design: Automated Test. Based on distinct software (2 independent) changes**
  - **to distinct modules, and**
  - **running reasonable test sets,**
  - **until further notice**
  - **or failure.**

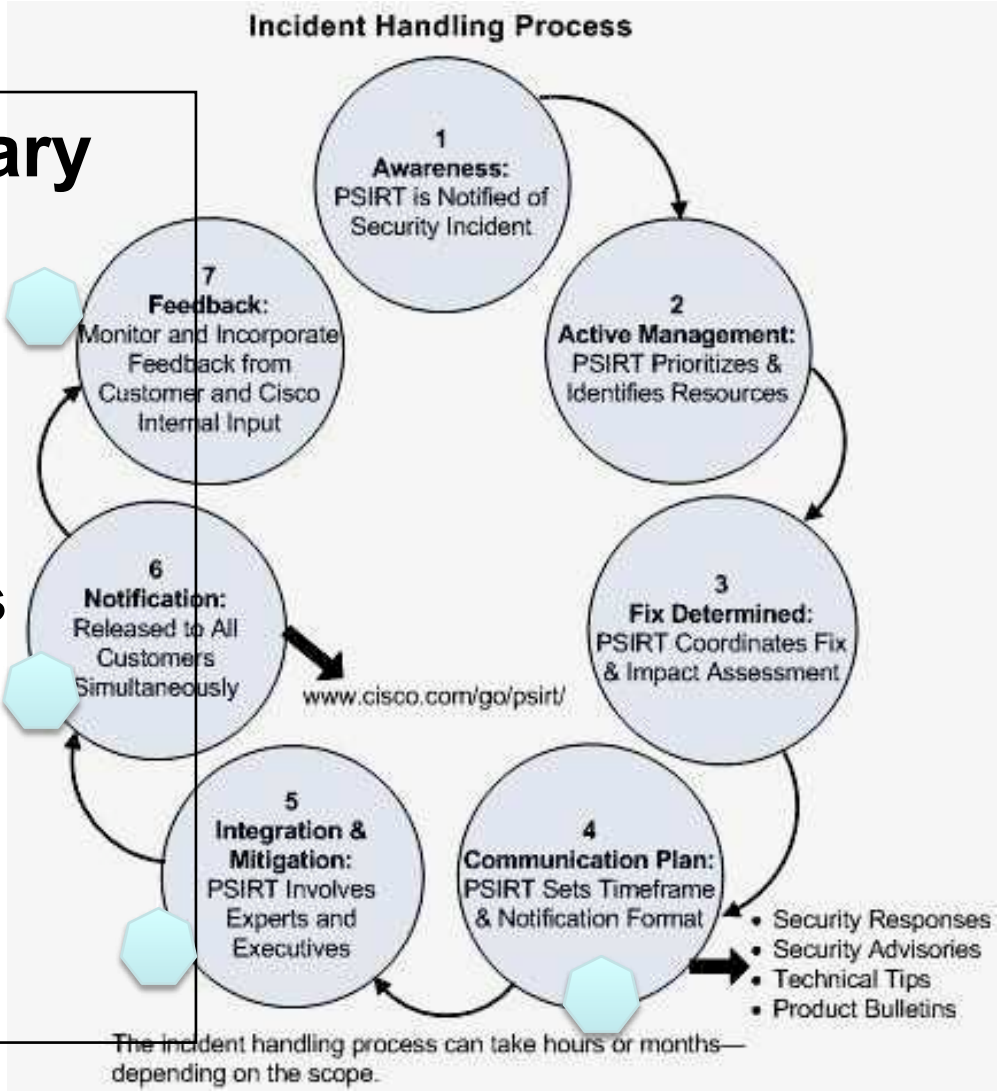
9. Field Pilot Test Time

- **Design:**
  - **After 30 minutes successful Local Test**
  - **the changes are implemented**
    - **at a customer pilot site**
      - **for more realistic testing,**
        - **in operation,**
        - **in distinct software safe mode.**

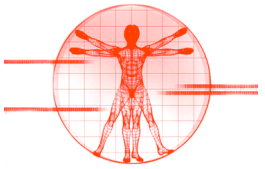
10. Change Distribution Time



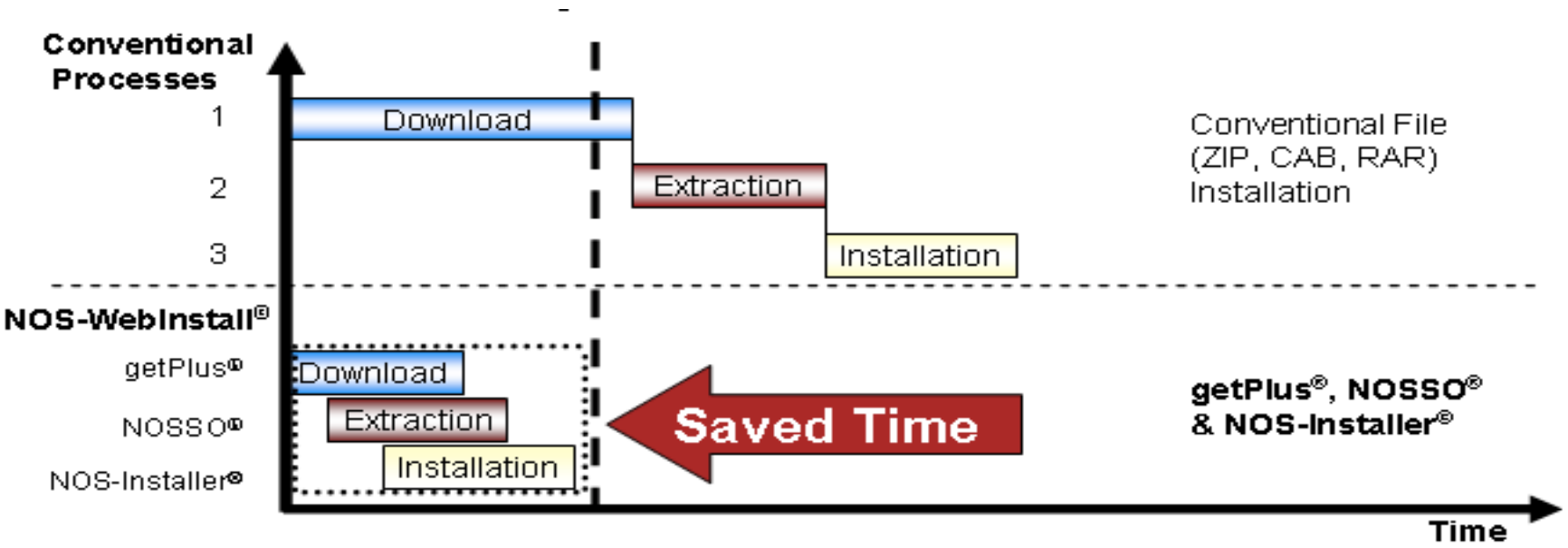
- **Design: All necessary changes are**
  - **readied and**
  - **uploaded for customer download,**
  - **even before Local Tests Begin,**
  - **and changed only**
    - **if tests fail.**



11. Customer Installation Time

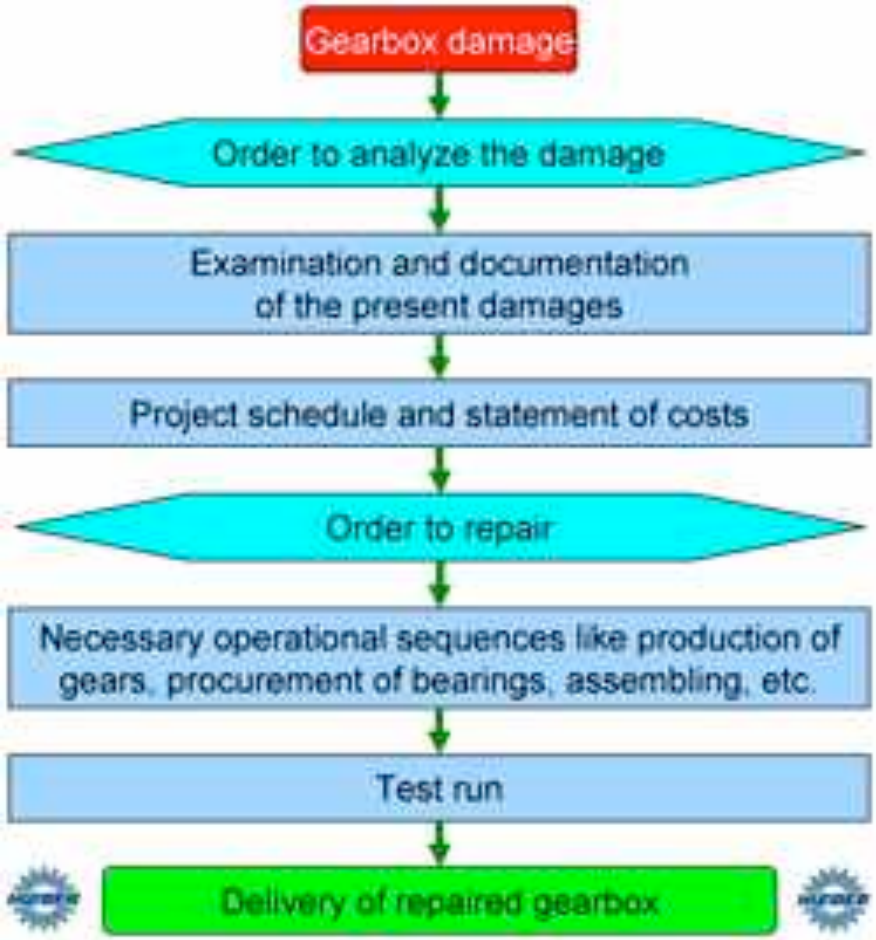


- **Design: Customer is given options of**
  - **manual or**
  - **automatic changes,**
  - **under given circumstances**



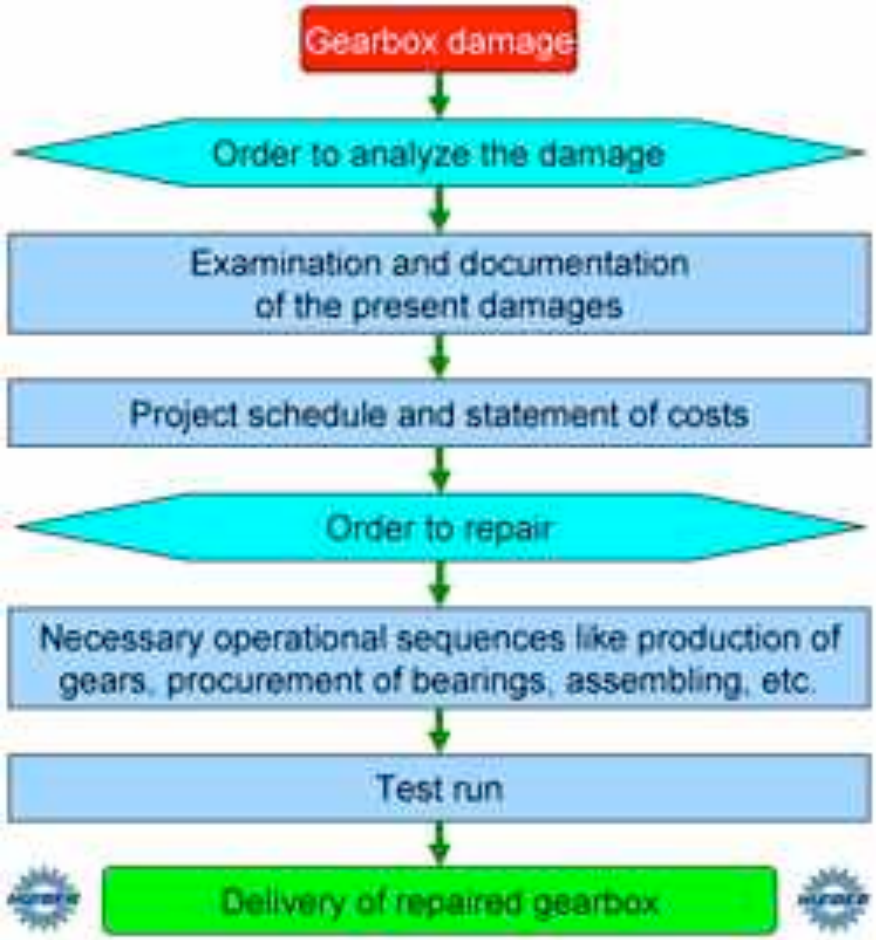
12. Customer Damage Analysis Time

- Design:
- <local customer solution>.
- We don't have good automation here.
- Assume none until proven otherwise.
- We need to be aware of
  - all reports sent
  - and databases updated that may need correction.
-



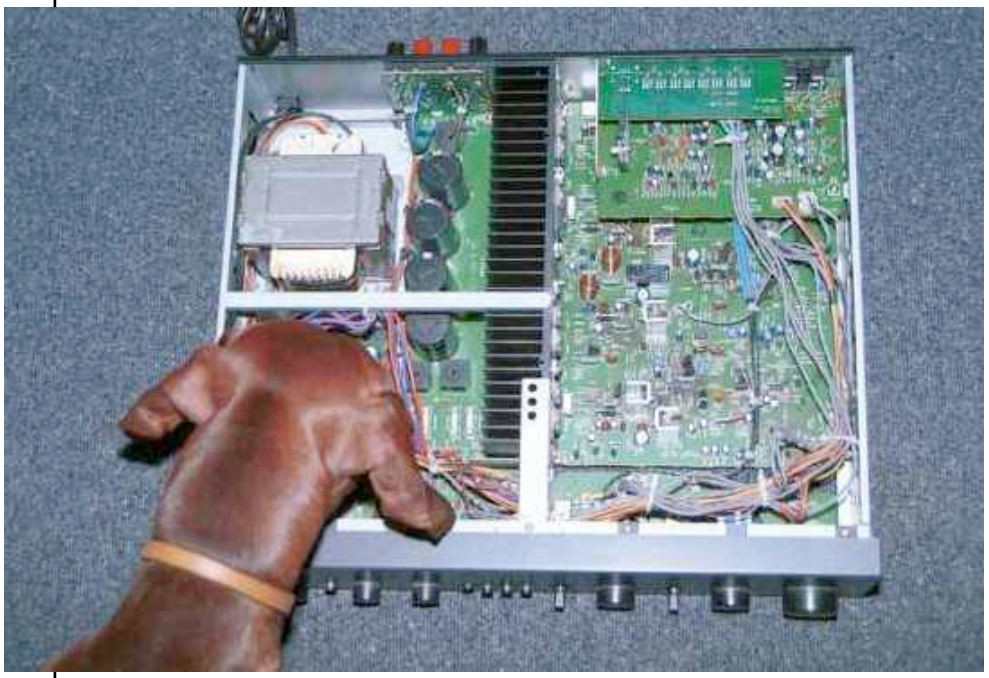
13. Customer-Level-Recovery Time

- **Design:**
- **same problem as Customer Damage Analysis Time**
- **may be highly local and manual.**
- **Is it really out of our control?**

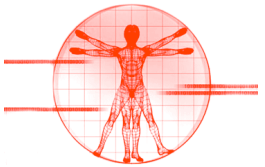


14. Customer QC of Recovery, Time.

- **Design:**
- **30-minute Quality Control**
  - **of recovery results,**
  - **assisted by our quality standards,**
  - **and *for critical customers***
  - **QC By our staff,**
    - **From our office**
    - **or on customer site.**



Main Point



Many Means

- My main point is
  - that each sub-process of the maintenance operation
  - tends to require a separate and distinct design (1 or more designs each).
- There is nothing simple
  - like software people seem to believe,
  - that better code structures,
  - coding practices, documentation,
  - and tools
  - will solve the maintenance problem.

Many Ends

Many Impacts

Design Ideas ->	*Technology Investment*	*Business Practices*	*People*	*Empowerment*	*Principles of IMA Management*	*Business Process Re-engineering*	*Sum Requirements*
Customer Service ? <->0 Violation of agreement	50%	10%	5%	5%	5%	60%	185%
Availability 90% <-> 99.5% Up time	50%	5%	5–10%	0%	0%	200%	265%
Usability 200 <-> 60 Requests by Users	50%	5–10%	5–10%	50%	0%	10%	130%
Responsiveness 70% <-> ECP's on time	50%	10%	90%	25%	5%	50%	180%
Productivity	45%	60%	35%	1[illegible]	[illegible]	53%	303%
[illegible]	50%	5%	[illegible]%	45%	[illegible]	61%	251%
[illegible]							
Data Integrity 88% <-> 97% Data Error %	42%	[illegible]%	25%	5%	70%	25%	177%
[illegible]nology Adaptability	5%	[illegible]%	[illegible]%	[illegible]	0%	[illegible]	160%
[illegible]	80%	20%	60%	75%	20%	5%	260%
Resource Adaptability 2.1M <-> ? Resource Change	10%	80%	5%	50%	50%	75%	270%
Cost Reduction FADS <-> 30% Total Funding	50%	40%	10%	40%	50%	[illegible]	[illegible]
Sum of Performance	*482%*	*280%*	*305%*	*390%*	*315%*	[illegible]	[illegible]
Money % of total budget	15%	4%	3%	4%	6%	[illegible]	[illegible]
Time % total work months/year	15%	15%	20%	10%	20%	18%	98%
Sum of Costs	*30*	*19*	*23*	*14*	*26*	*22*	
Performance to Cost Ratio	*16:1*	*14:7*	*13:3*	*27:9*	*12:1*	*29:5*	

Next Slide!

DoDef. Persinscom Impact Estimation Table:

Designs

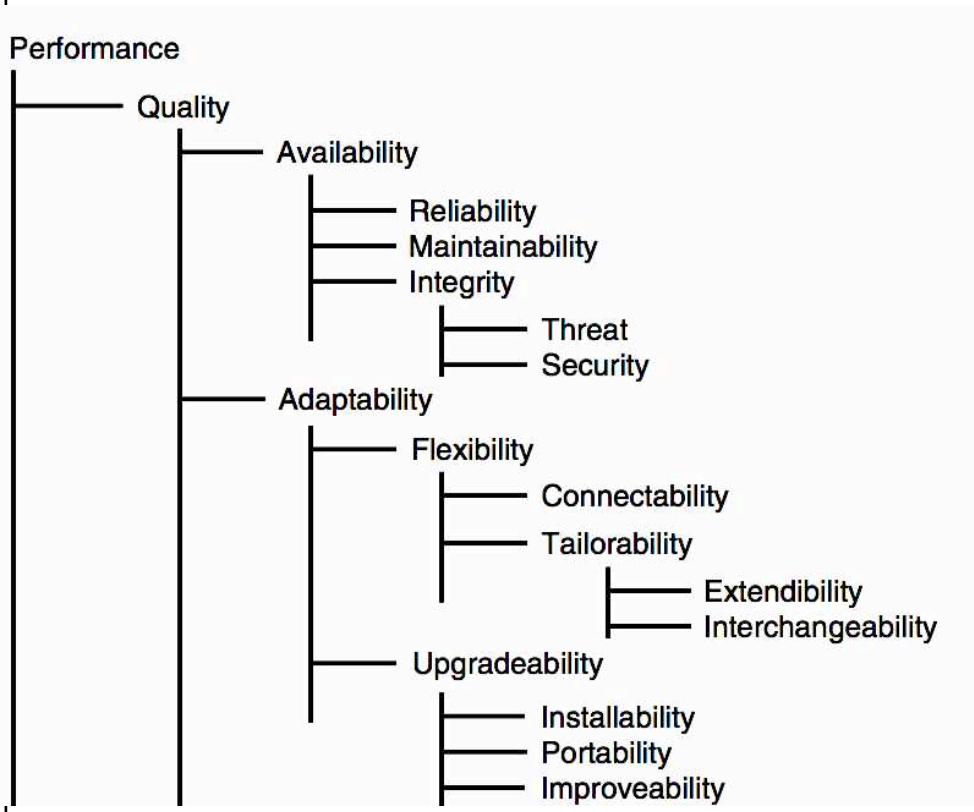
Requirements

R→ D Impacts

Design Ideas ->	*Technology Investment*	*Business Practices*	*People*	*Empowerments*	*Principles of IMA Management*	*Business Process Re-engineering*	*Sum Requirements*
	50%	10%	5%	5%	5%	60%	185%
Availability 90% <-> 99.5% Up time	50%	5%	5–10%	0%	0%	200%	265%
Usability 200 <-> 60 Requests by Users	50%	5–10%	5–10%	50%	0%	10%	130%
Responsiveness 70% <-> ECP's on time	50%	10%	90%	25%	5%	50%	180%
Productivity	45%				100%	53%	303%
3:1 Return on Investment	50%				15%	61%	251%
Morale 72 <-> 60 per month on Sick Leave							
Data Integrity 88% <-> 97% Data Error %	42%	10%	25%	5%	70%	25%	177%
Technology Adaptability 75% Adapt Technology	5%	30%	5%	60%	0%	60%	160%
Requirement Adaptability ? <-> 2.6% Adapt to Change	80%	20%	60%	75%	20%	5%	260%
Resource Adaptability 2.1M <-> ? Resource Change	10%	80%	5%	50%	50%	75%	270%
Cost Reduction FADS <-> 30% Total Funding	50%	40%	10%	40%	50%	50%	240%
Sum of Performance	*482%*	*280%*	*305%*	*390%*	*315%*	*649%*	
Money % of total budget	15%	4%	3%	4%	6%	4%	36%
Time % total work months/year	15%	15%	20%	10%	20%	18%	98%
Sum of Costs	*30*	*19*	*23*	*14*	*26*	*22*	
Performance to Cost Ratio	*16:1*	*14:7*	*13:3*	*27:9*	*12:1*	*29:5*	

Broader Maintainability Concepts

- Maintainability in the strict engineering sense is usually taken to mean **bug fixing.**
- I have however been using it *thus far* to describe ***any software change activity or process.***
- We could perhaps better call it **'software change ability'.**
- Different classes of change, will have different requirements related to them,
  - and consequently **different technical solutions**.
- It is important that we be very clear
  - in setting requirements,
  - and doing corresponding design,
  - exactly what **types of change** we are talking about.



General 'Change Attribute' **Tailoring** Slide 155!

- The following slides will give a **general set of patterns** for
  - defining and distinguishing *different* **classes** of 'maintenance'.
- But in your *real* world, you will want to **tailor** the definitions to *your* domain.
  - You can initially tailor using the '**Scale**' of measure definition.
  - And continued tailoring can be done by defining **[conditions]** in the requirement level qualifier.

Scale:
% of transactions successfully completed by defined [Person] doing defined [Task].

Goal [Task = Update, Person = New Hire, Deadline = Phase 3]
60%

A generic set of performance measures, including several related to change.

For example:

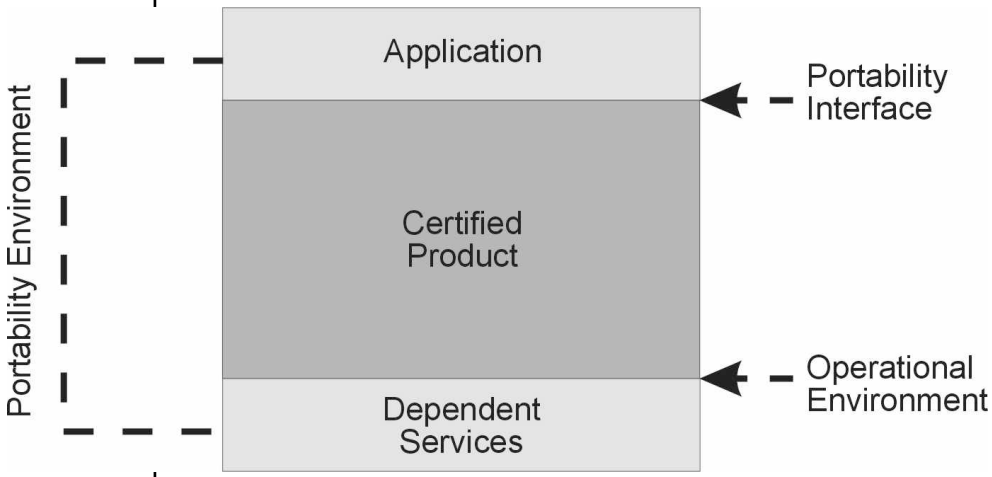
Code Portability:

Scale:

Effort in Hours
needed to Port
each 1000 Non-Commentary Lines of Code
from a defined [Home Environment]
to a defined [Target Environment],
using defined [Tools]
and defined [Personnel].

Goal

[Home Environment = {.net, Oracle,} ,
Target Environment = {Java++, Open Source, Linux},
Tools = Convert Open ,
Personnel = {Experienced Experts, India}] 60 hours.



A Generic Set of Performance measures – including several related to 'change'

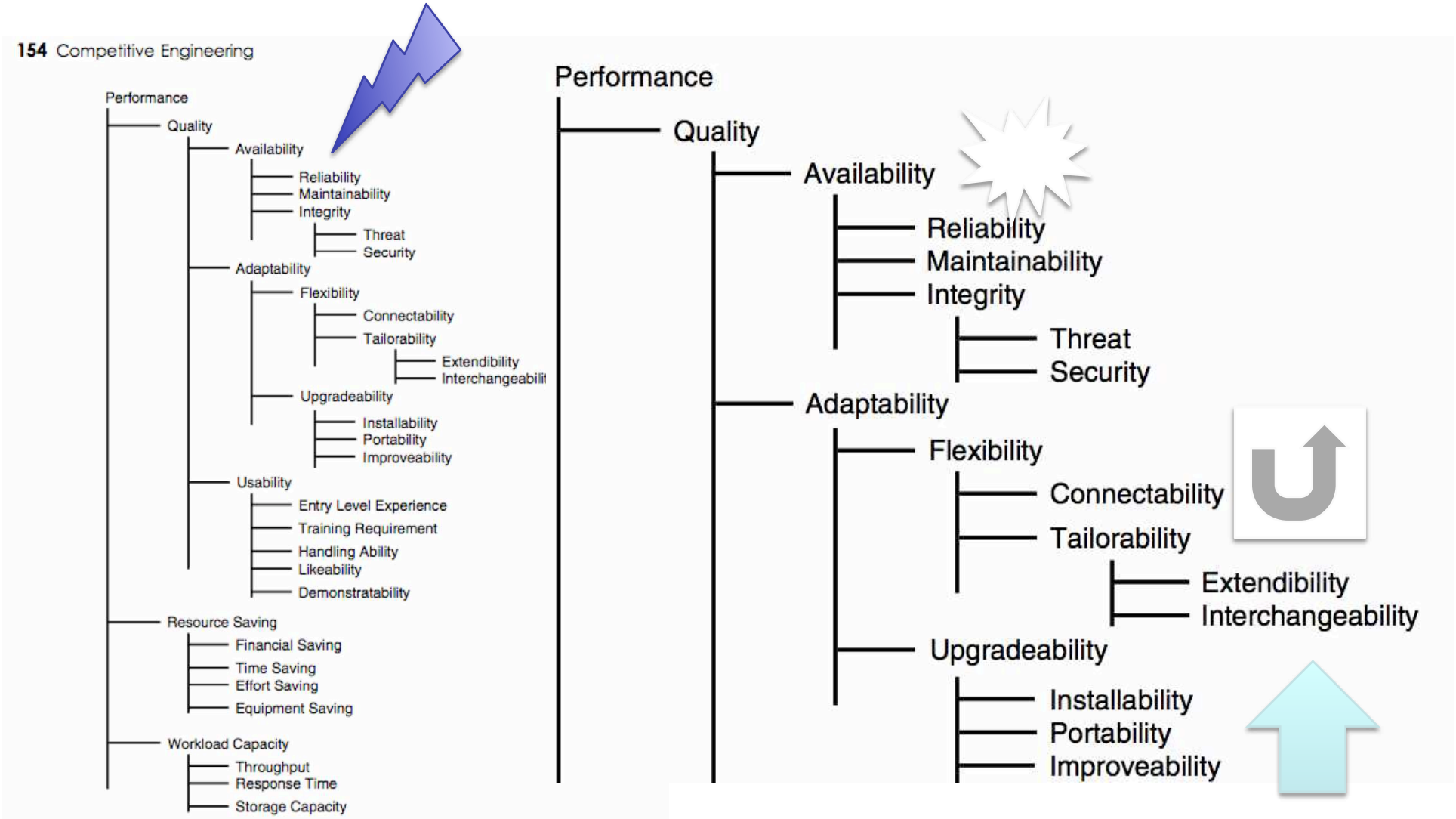
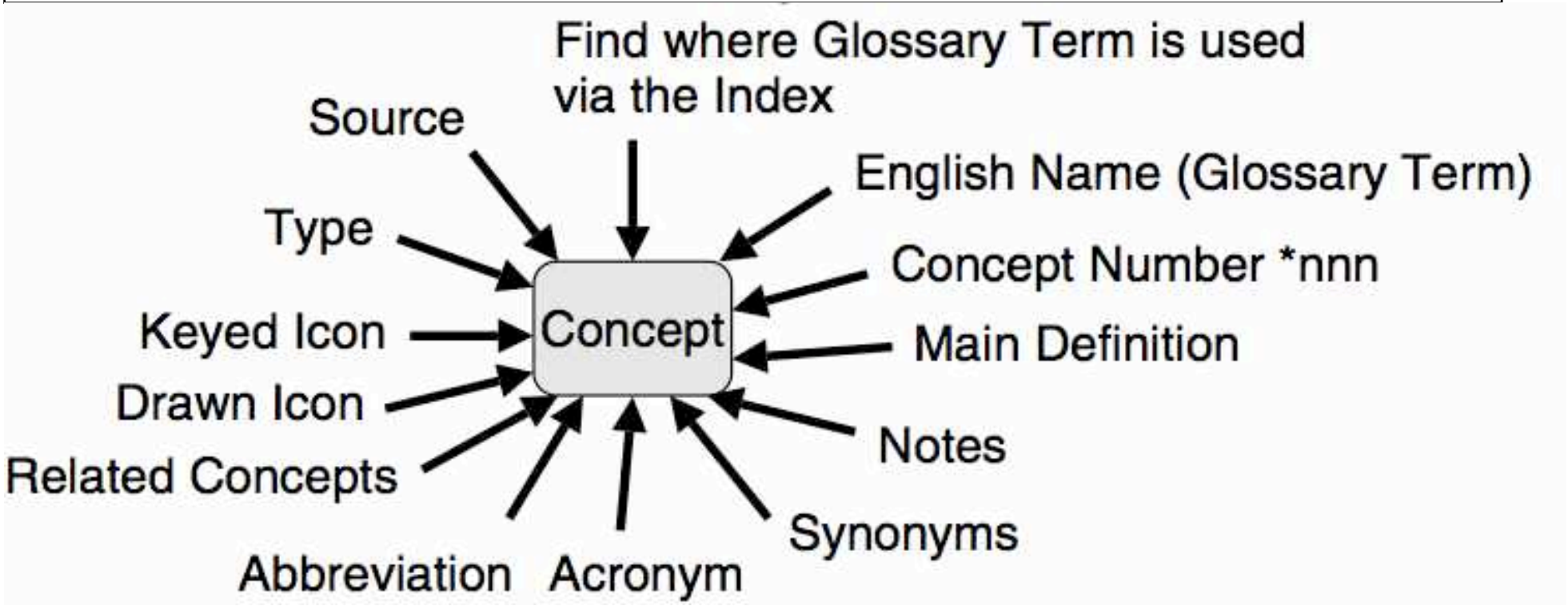


Figure 5.3
One decomposition possibility for performance attributes with emphasis on the detail of the quality attributes.

The *attribute names* used are arbitrary choices by the author.

- They only start to take on meaning when defined,
  - with a Scale of measure.
- There are no accepted or acceptable standards here,
  - and certainly not for software.
  - Even in hardware engineering, there is an accepted **pattern** – such as "Scale: Mean Time to Repair".
  - But it is accepted that we have to *further* define such concepts *locally*,
    - such as the meaning of 'Repair'.



Maintainability Measures

- ! Here are some of the general **patterns** we can use to define and distinguish the different classes of change processes on software.
- ! First the ‘Bug Fixing’ pattern (from which we derived the example at the beginning of this talk).

Maintainability:
Type: Complex Quality Requirement.
Includes: {Problem Recognition, Administrative Delay, Tool Collection, Problem Analysis, Change Specification, Quality Control, Modification Implementation, Modification Testing {Unit Testing, Integration Testing, Beta Testing, System Testing}, Recovery}.

Problem Recognition:
Scale: Clock hours from defined [Fault Occurrence: Default: Bug occurs in any use or test of system] until fault officially recognized by defined [Recognition Act: Default: Fault is logged electronically].

Administrative Delay:
Scale: Clock hours from defined [Recognition Act] until defined [Correction Action] initiated and assigned to a defined [Maintenance Instance].

Tool Collection:
Scale: Clock hours for defined [Maintenance Instance: Default: Whoever is assigned] to acquire all defined [Tools: Default: all systems and information necessary to analyze, correct and quality control the correction].

Problem Analysis:
Scale: Clock time for the assigned defined [Maintenance Instance] to analyze the fault symptoms and be able to begin to formulate a correction hypothesis.

Change Specification:
Scale: Clock hours needed by defined [Maintenance Instance] to fully and correctly describe the necessary correction actions, according to current applicable standards for this.
Note: This includes any additional time for corrections after quality control and tests.

Quality Control:
Scale: Clock hours for quality control of the correction hypothesis (against relevant standards).

Modification Implementation:
Scale: Clock hours to carry out the correction activity as planned. "Includes any necessary corrections as a result of quality control or testing."

Modification Testing:

Unit Testing:
Scale: Clock hours to carry out defined [Unit Test] for the fault correction.

Integration Testing:
Scale: Clock hours to carry out defined [Integration Test] for the fault correction.

Beta Testing:
Scale: Clock hours to carry out defined [Beta Test] for the fault correction before official release of the correction is permitted.

System Testing:
Scale: Clock hours to carry out defined [System Test] for the fault correction.

Recovery:
Scale: Clock hours for defined [User Type] to return system to the state it was in prior to the fault and, to a state ready to continue with work.

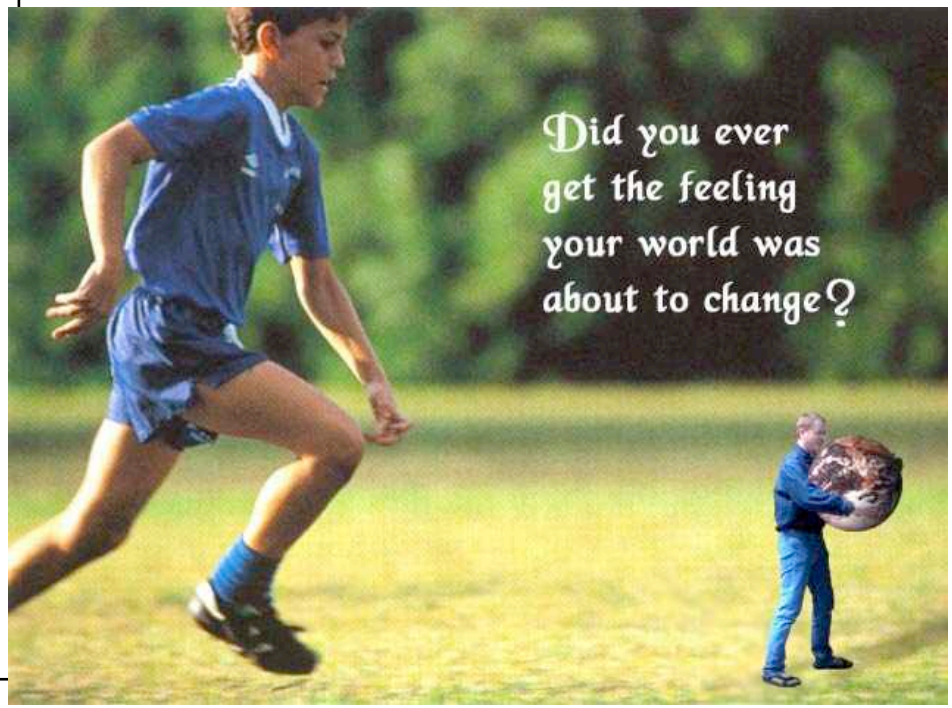
Source: The above is an extension of some basic ideas from Ireson, Editor, Reliability Handbook, *McGraw-Hill, 1966 (#8 on p. 966).*

Figure 5.4

Maintainability components, derived from a hardware engineering view, adopted for software.

Notice that *Maintainability* in the narrow sense (fix bugs) is quite separate from other 'Adaptability' concepts.

- **This is normal *engineering*,**
  - **Which places fault repair together with reliability and availability;**
  - **Those 3 determine the *immediate* operational characteristics of the system.**
- **The other forms of adaptability are more about *potential* future upgrades to the system,**
  - ***change*, rather than *repair*.**
- **Change and repair, have in common that**
  - **our system *architecture* has to make it easy to change, analyze and test.**
- **The system *itself* is unaware of**
  - **whether we are *correcting* a *fault***
  - **or *improving* the system.**
- **The consequence is that**
  - ***much of the maintenance-impacting 'design' or 'architecture'***
  - **benefits**
  - ***most of the types of maintenance (fix and adapt).***



Here are a *generic* set of definitions for the '*Adaptability*' concepts.

Adaptability: 'The **efficiency** with which a system can be changed.'

Gist: Adaptability is a measure of a system's ability to change.

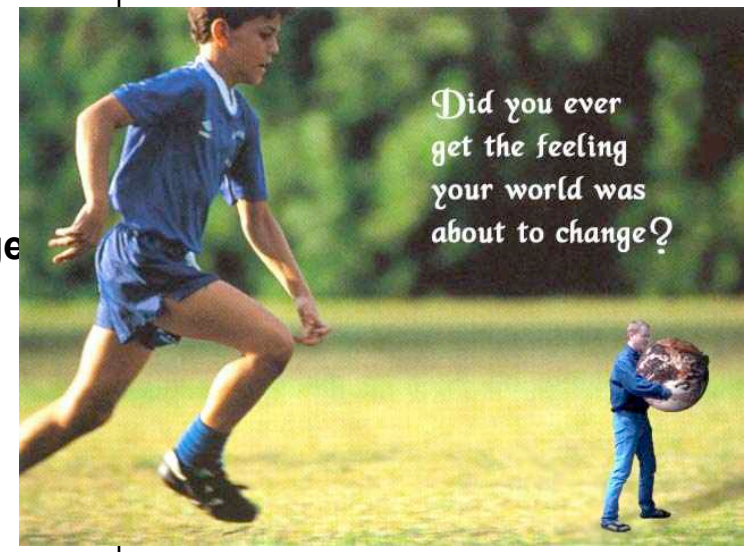
Includes: { a set of scalar variables, such as Portability}.

Note: probably not `simple` enough to define with a **single** Scale.

Type: *Complex* Quality Attribute.

Since,

- **if given sufficient resource, a system can be change**
  - **almost any way,**
- **the primary concern is with the amount of**
  - **resources**
    - **(such as time, people, tools and finance)**
- **needed to bring about specific changes**
  - **(the change 'cost').**



The Adaptive Cycle

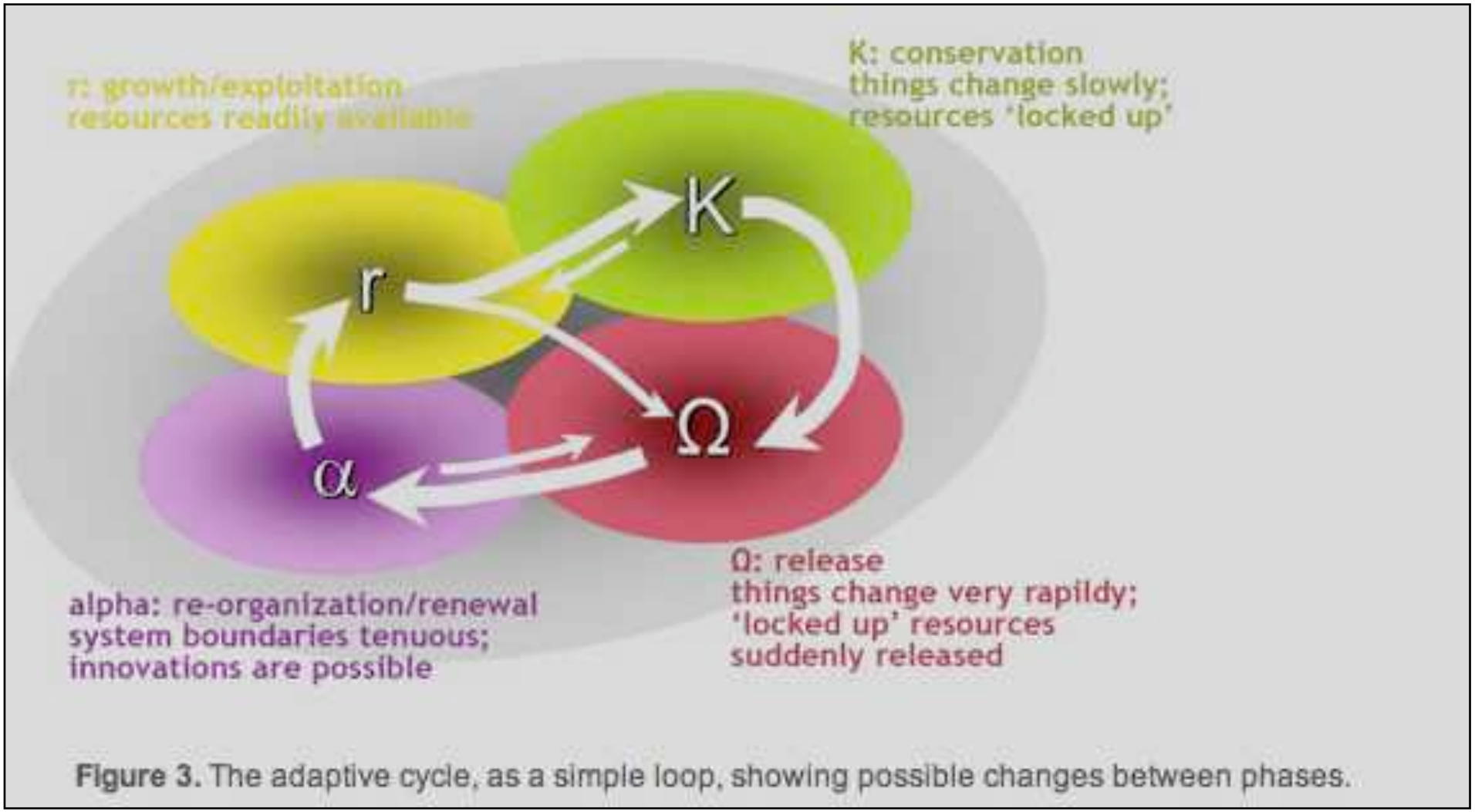


Figure 3. The adaptive cycle, as a simple loop, showing possible changes between phases.

http://www.resalliance.org/564.php!

Adaptability:
Viewed as
Elementary or ***Complex*** concept..

Adaptability:

Type: **Elementary** Quality Requirement.

Scale: Time needed to adapt a defined [System] from a defined [**Initial State**] to another defined [**Final State**] using defined [**Means**].

Adaptability:

Type: **Complex** Quality Requirement.

Includes*: {Flexibility, Upgradeability}.*

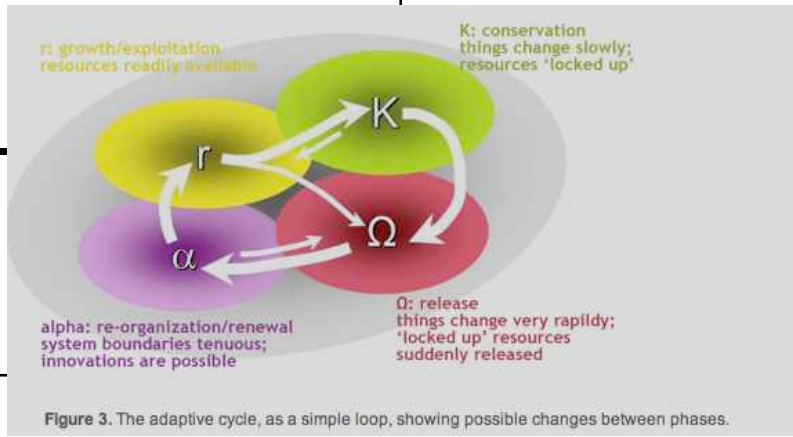


Figure 3. The adaptive cycle, as a simple loop, showing possible changes between phases.

“No system can be understood or managed by focusing on it at a *single* scale.”

Multiple scales and cross-scale effects - "Panarchy"

No system can be understood or managed by focusing on it at a single scale.

- **All systems (and SESs especially) exist and function at multiple scales of space, time and social organization,**
  - **and the interactions across scales are fundamentally important in determining the dynamics of the system at any particular focal scale.**
  - **This interacting set of hierarchically structured scales has been termed a "panarchy" (Gunderson and Holling 2003).**

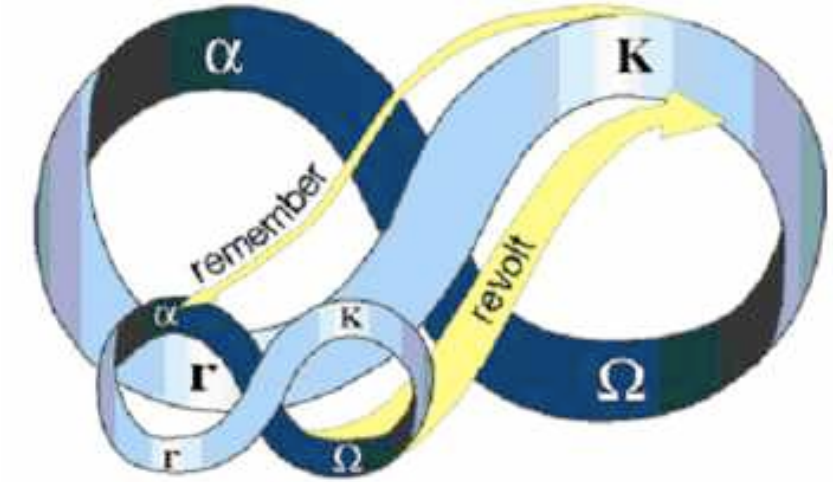


Figure 4. "Panarchy" - nested adaptive cycles, with influences between scales.

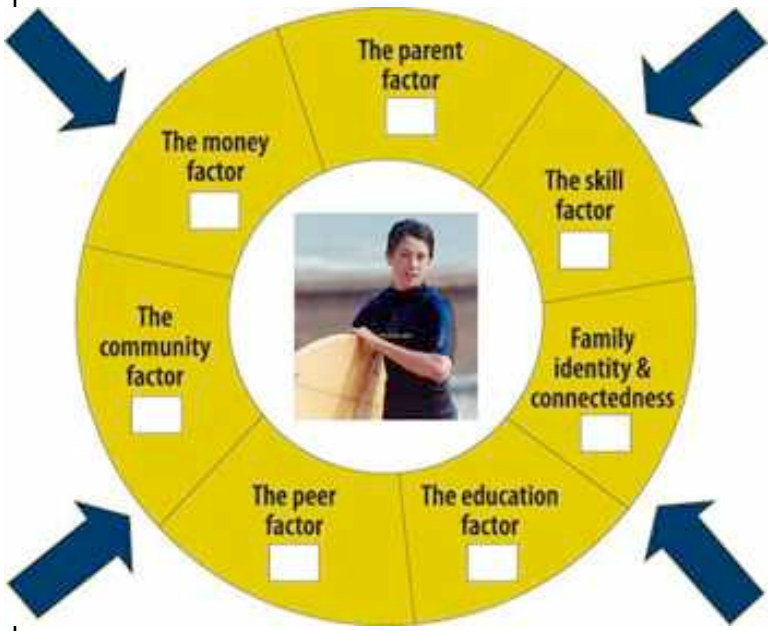
Flexibility:

Gist: 'Flexibility' concerns the
'in-built' ability of the system
to adapt,
or to be adapted,
by its users,
to suit conditions
(without any fundamental system modification
by system development).

Type: Complex Quality Requirement.

Includes: {Connectability, Tailorability}.
See next 2 slides!

Possible Synonyms: Resilience, Robustness



Connectability:

‘The cost to interconnect the system to its environment.’

Gist: The cost of connecting **one set of interfaces** to defined **environments** with **other interfaces**

Part Of: Flexibility.

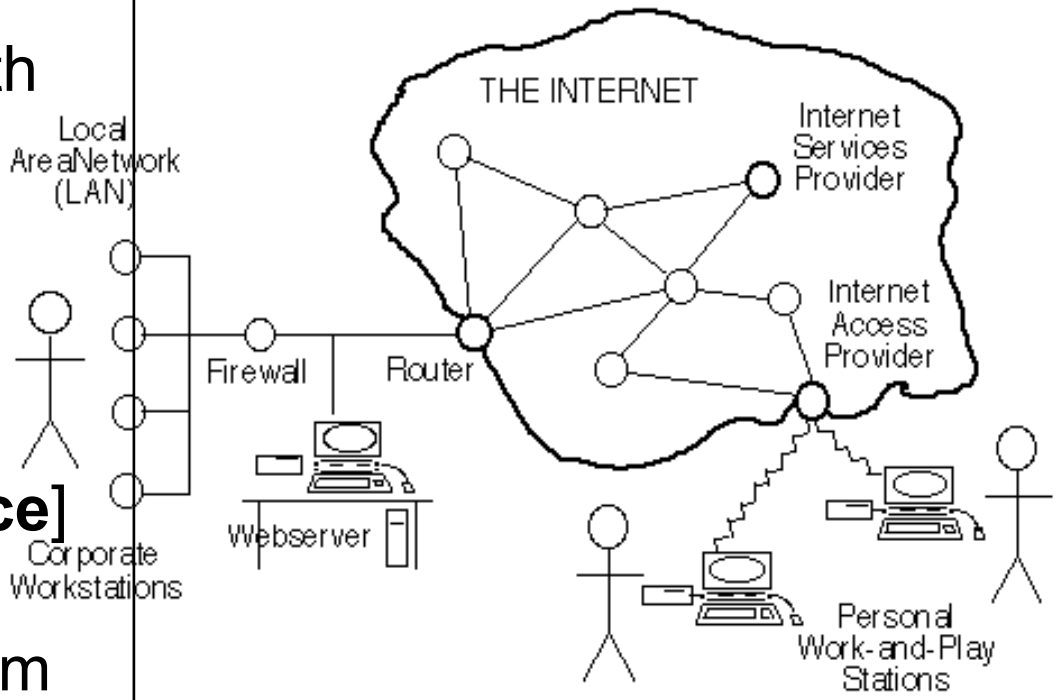
Scale: the **Effort** needed

to connect a defined [**Home Interface**]

to a defined [**Target Interface**]

using defined [**Methods**]

with minimum allowed system [**Degradation**].



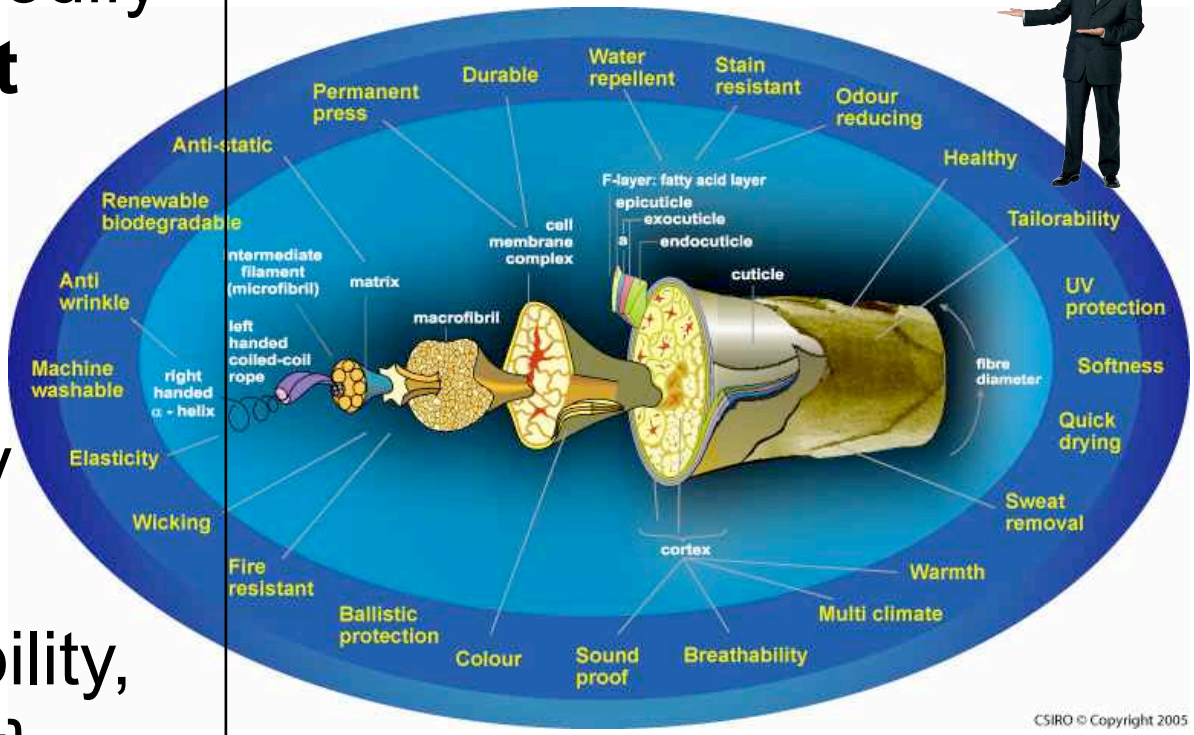
Tailorability:

Gist: The **cost** to modify the system to **suit** defined **future** conditions.

Part Of: Flexibility.

Type: *Complex* Quality Requirement.

Includes: {Extendibility, Interchangeability}.



Multiple Attributes of Wool Fiber !!

Extendibility: Scalability

Extendibility:
Part Of: Tailorability.
Synonym: Scalability.
Scale: The **cost** to add to
a defined [**System**]
a defined [**Extension Class**]
and defined [**Extension Quantity**]
using a defined [**Extension Means**].

"In other words, add such things as a new user or a new node."

Type: *Complex* Quality Attribute.

Includes: {Node Addability, Connection Addability, Application Addability, Subscriber Addability}.

Interchangeability:

‘The cost to modify use of system components.’

Interchangeability

Gist: This is concerned with the ability to modify the system, to switch from using a certain set of system components, to using another set.

Part Of: Tailorability.

Type: `Elementary` Quality Attribute.

“For example, this could be a daily occurrence switching system mode from day to night use.”

Scale: the Effort needed to
Successfully,
without Intolerable Side Effects,
replace a defined [Initial Set] of components,
with a defined [Replacement Set] of system components,
using defined [Means].

Upgradeability:

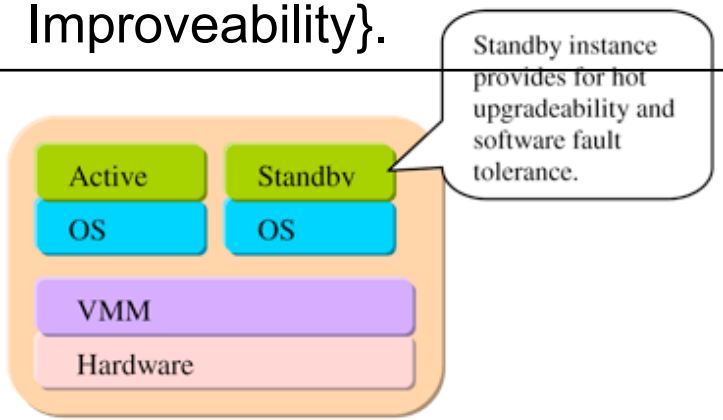
‘The cost to modify the system fundamentally;
either to install it, or to change out system components.’

Upgradeability:

Gist: This concerns the ability of the system to be modified by the system developers or system support in planned stages (as opposed to unplanned maintenance or tailoring the system).

Type: *Complex* Quality Requirement.

Includes: {Installability, Portability, Improveability}.



Installability: ‘The cost to install in defined conditions.’

Pattern: This concerns installing the system code and also, installing it in new locations to extend the system coverage. Could include conditions such as the installation being carried out by a customer or, by an IT professional on-site.

Portability: ‘The cost to move from location to location.’

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

Type: Complex Quality Requirement.

Includes: {Data Portability,
Logic Portability,
Command Portability,
Media Portability}.

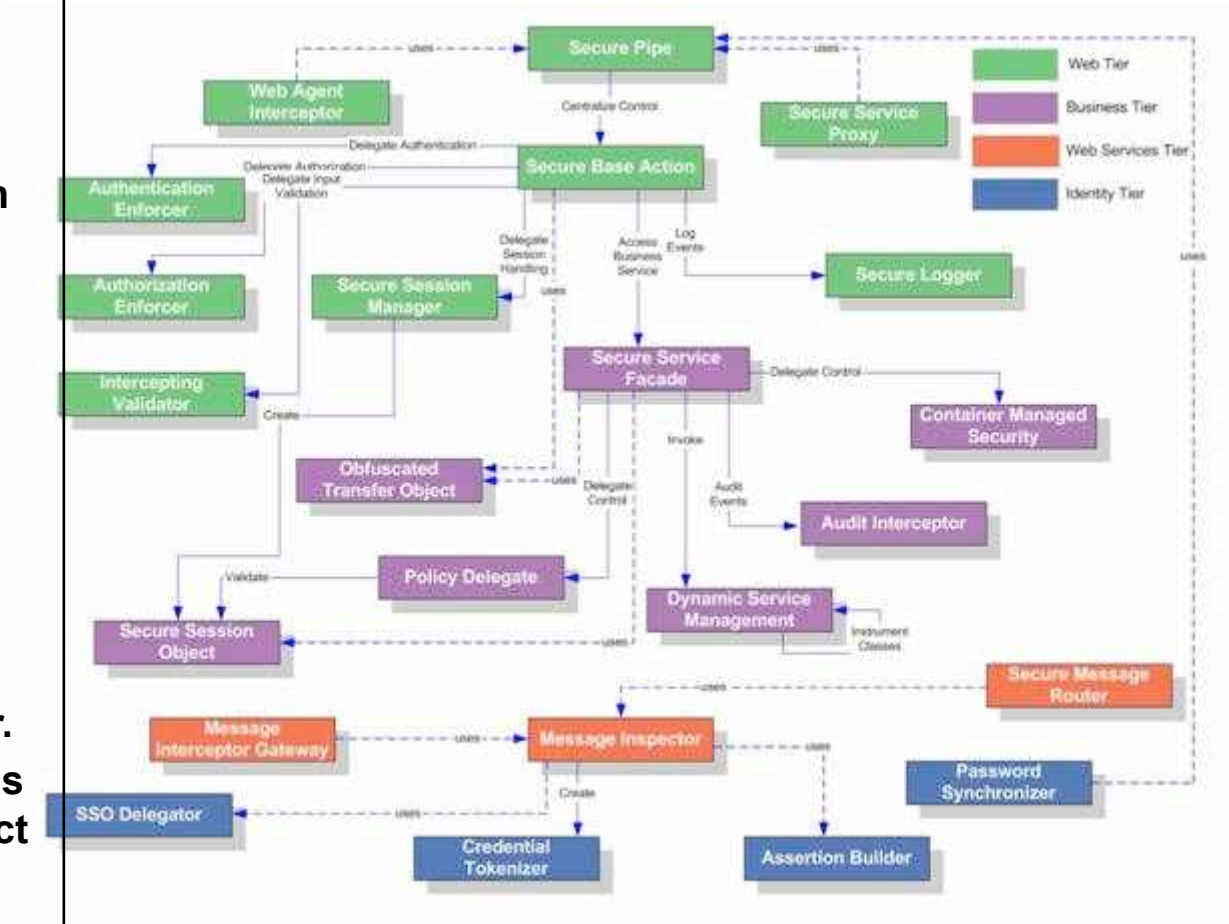
Improveability: ‘The cost to enhance the system.’

Gist: The ability to replace system components with others, which possesses improved (function, performance, cost and/or design) attributes.

Scale: The cost to add to a defined [System] a defined [Improvement] using a defined [Means].

This Basic 'Adaptability' Pattern Was Successfully Applied

- Hopefully this set of patterns
  - gives you a departure point
  - for defining those maintenance attributes
  - you might want to control, quantitatively.
- The above adaptability definition
  - was use to co-ordinate the work
    - of 5,000 software engineers,
    - and 5,000 hardware engineers,
    - in UK,
    - in bringing out a new product line at a computer manufacturer.
    - Where 'Adaptability' was the Number One Product Characteristic
  - The Company became profitable for the next 14 years..



Security Patterns!

The Software Architect Role in Maintainability

The role of the software architect is:

- to participate in **clarification of the requirements** that will be used as inputs to their architecture process.
- to insist that the requirements are **testably clear**: that means with defined and agreed scales of measure, and defined required levels o performance.
- to then **discover appropriate architecture**,
  - –! capable of delivering those levels of performance, hopefully within resource constraints, and
- **estimate** the probable **impact** of the architecture,
  - –! on the requirements (Impact Estimation)
- **define** the architecture in such **detail**
  - –! that the intent **cannot be misunderstood** by implementers,
  - –! and the desired **effects** are bound to be **delivered**.
- **monitor** the developing system as the architecture is applied in practice,
- and **make** necessary **adjustments**.
- finally **monitor** the **performance characteristics** throughout the lifetime of the system,
  - –! and make necessary **adjustments** to requirements
  - –! and to architecture,
  - –! in order to **maintain** needed system **performance** characteristics.

Evaluating Maintainability Designs Using Impact Estimation

	A	B	C	D	E	F	G	BX	BY	BZ	CA
1											
2		Current Status	Improvements		Goals			Step9			
3								Recoding			
4								Estimated impact		Actual impact	
5		Units	Units	%	Past	Tolerable	Goal	Units	%	Units	%
6					Usability.Replacability (feature count)						
7		1,00	1,0	50,0	2	1	0				
8					Usability.Speed.NewFeaturesImpact (%)						
9		5,00	5,0	100,0	0	15	5				
10		10,00	10,0	200,0	0	15	5				
11		0,00	0,0	0,0	0	30	10				
12					Usability.Intuitiveness (%)						
13		0,00	0,0	0,0	0	60	80				
14					Usability.Productivity (minutes)						
15		20,00	45,0	112,5	65	35	25	20,00	50,00	38,00	95,00
20					Development resources						
21			101,0	91,8	0		110	4,00	3,64	4,00	3,64

- ! *See Powerpoint Notes for detailed written comment.*
- !

Architecture Level Impact Estimation Table

		.....Deliverables						
		Telephony	Modularity	Tools	User Experience	GUI & Graphics	Security	Enterprise
Business Objective								
Time to Market		10%	10%	15%	0%	0%	0%	5%
Product Range		0%	30%	5%	10%	5%	5%	0%
Platform Technology		10%	0%	0%	5%	0%	10%	5%
Units		15%	5%	5%	0%	0%	10%	10%
Operator Preference		10%	5%	5%	10%	10%	20%	10%
Commoditization		10%	-20%	15%	0%	0%	5%	5%
Duplication		10%	0%	0%	0%	0%	5%	5%
Competitiveness		15%	10%	10%	10%	20%	10%	10%
User Experience		0%	20%	0%	30%	10%	0%	0%
Downstream Cost Saving		5%	10%	0%	10%	0%	0%	5%
Other Country		5%	10%	0%	10%	5%	0%	0%
Total Contribution		90%	80%	55%	85%	50%	65%	55%
Cost (£M)		0.49	1.92	0.81	1.21	2.68	0.79	0.60
Contribution to Cost Ratio		**184**	42	68	70	19	82	92

•! See PPT Notes

Engineering "Maintainability": Green Week
Weekly 'Refactoring' at Confirmit

Current Status		Improvement	Goals			Step 6 (week 14)		Step 7 (week 15)	
	Units		Past	Tolerable	Goal	Estimated Impact	Actual Impact	Estimated Impact	Actual Impact
	100,0	100,0	0	80	100			100	100
Speed									
	100,0	100,0	0	80	100	100	100		
Maintainability.Doc.Code									
	100,0	100,0	0	80	100	100	100		
InterviewerConsole									
NUnitTests									
	0,0	0,0	0	90	100				
PeerTests									
	100,0	100,0	0	90	100			100	100
FxCop									
	0,0	10,0	10	0	0				
TestDirectorTests									
	100,0	100,0	0	90	100			100	100
Robustness.Correctness									
	2,0	2,0	0	1	2	2	2		
Robustness.BoundaryConditions									
	0,0	0,0	0	80	100				
Speed									
	0,0	0,0	0	80	100				
ResourceUsage.CPU									
	100,0	0,0	100	80	70	70			
Maintainability.Doc.Code									
	100,0	100,0	0	80	100	100	100		
SynchronizationStatus									
NUnitTests									

Speed

Maintainability

Nunit Tests

PeerTests

TestDirectorTests

Robustness.Correctness

Robustness.Boundary Conditions

ResourceUsage.CPU

Maintainability.DocCode

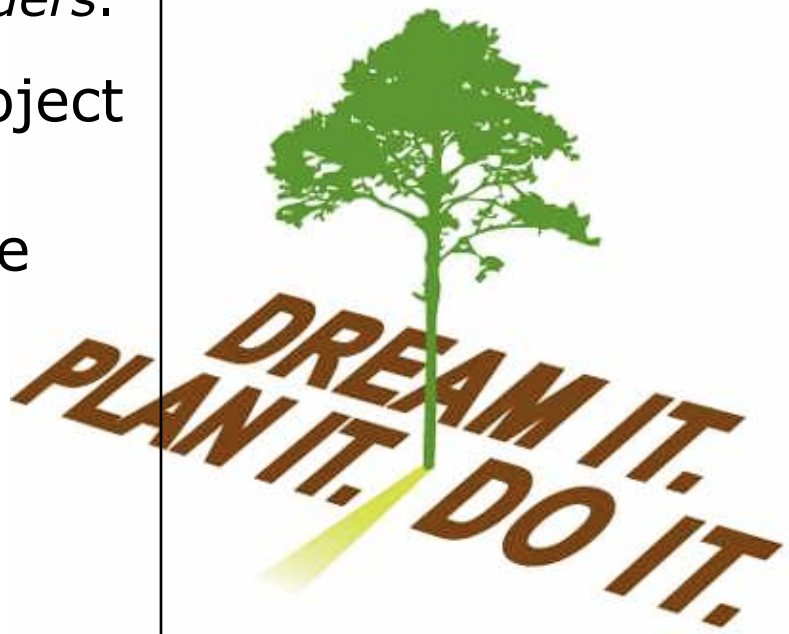
SynchronizationStatus

- Value Driven Planning

Value Driven Planning: 10 Value Principles

Value Driven Planning:
Stakeholders, Value Focus, Quantified, Stepwise

- Value Driven Planning focuses on
  - the primary values of key *stakeholders*.
- The *technology* used, and the project *processes* used are sub-ordinate.
- The critical stakeholder values are *quantified* and *trackable*.
- There is an assumption of
  - step by step achievement,
  - of *learning* at each step
  - and consequent *action*
    - to resolve problems of value achievement.

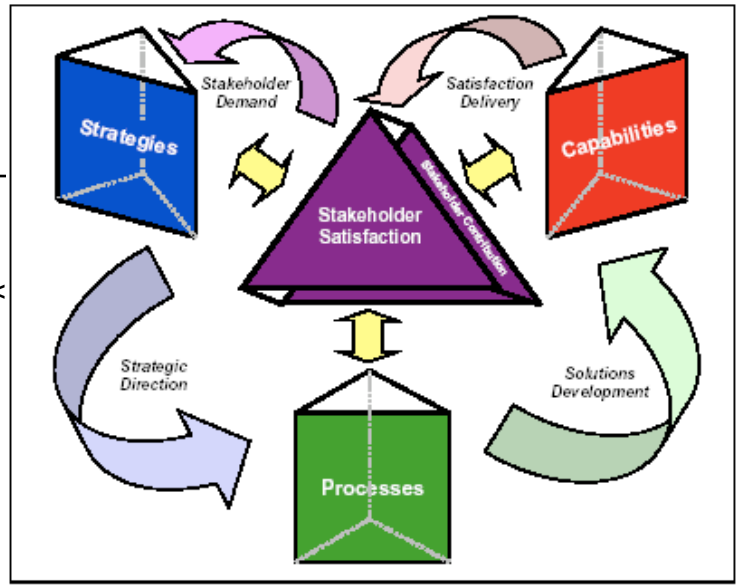


Gilb's 'Value Driven Planning' Principles:

1. **Critical Stakeholders determine the values**
2. **Values can and must be quantified**
3. **Values are supported by Value Architecture**
4. **Value levels are determined by timing, architecture effect, and resources**
5. **Value levels can differ for different scopes (where, who)**
6. **Value can be delivered early**
7. **Value can be locked in incrementally**
8. **New Values can be discovered (external news, experience)**
9. **Values can be evaluated as a function of architecture (Impact Estimation)**
10. **Value delivery will attract resources.**

Value Driven Planning Principles in Detail:

1. Critical Stakeholders determine the values

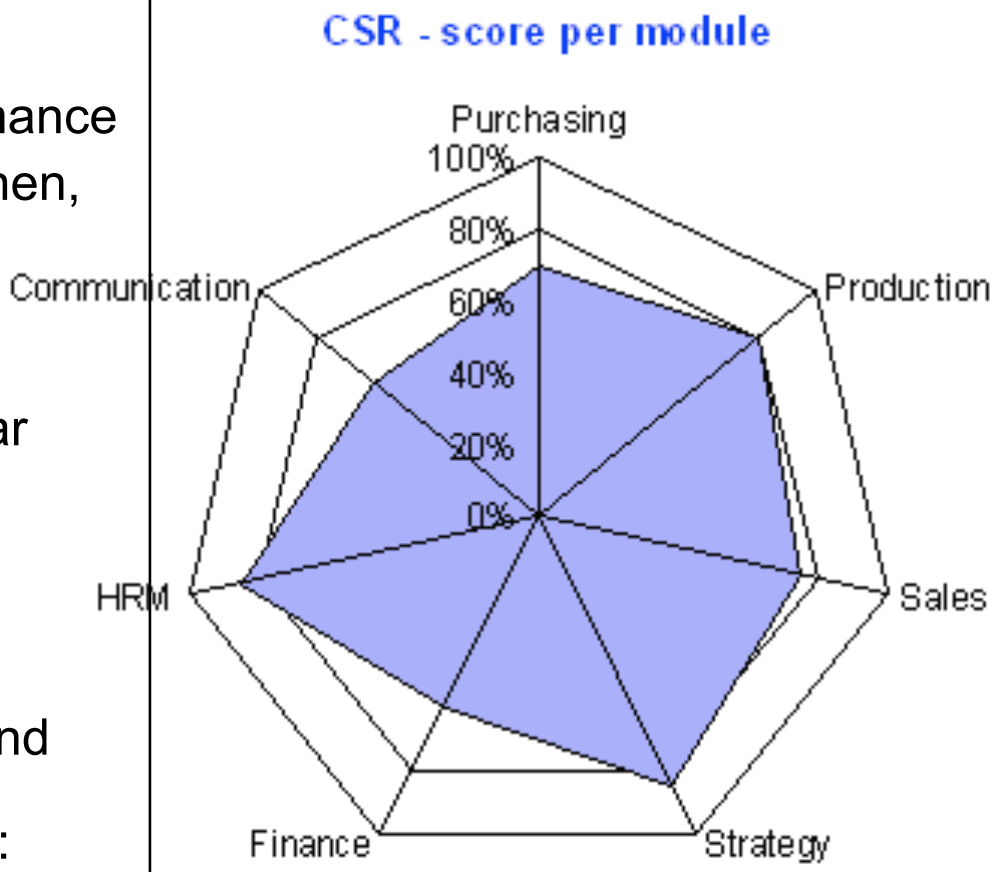


Critical: "having a decisive or crucial importance in the success or failure of something" < Dictionary

- The primary and prioritized values we need to deliver are determined by
  - analysis of the needs and values of stakeholders
    - stakeholders who can determine whether we *succeed* or *fail*.
- We cannot afford to satisfy *other (less critical)* levels, at other times and places, yet.
  - Because that might undermine our ability to satisfy the more critical stakeholders –
  - and consequently threaten our overall project success.

2. 'Values' can and must be *quantified*

- Values can, if you want, be expressed numerically.
  - With a defined scale of measure
  - with a deliverable level of performance
  - and with qualifier info [Where, When, If]
- Quantification is useful:
  - to clarify your own thoughts
  - to get real agreement to one clear idea
  - to allow for varied targets and constraints
  - to allow direct comparison with benchmarks
  - to put in Request for bids, bids and contracts
  - to manage project evolutionarily : track progress
  - as a basis for measurement and testing
  - to enable research on methods



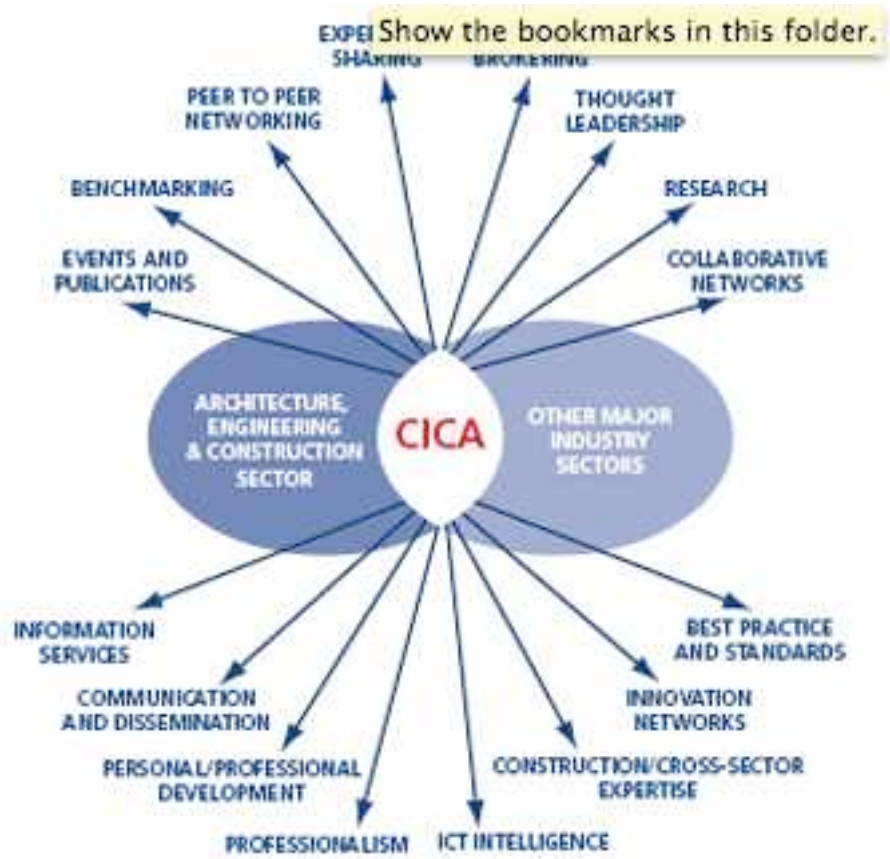
•Figure 1: Real (NON-CONFIDENTIAL version) example of an initial draft of setting the objectives that engineering processes must meet.

Business objective	Measure	Goal (200X)	Stretch goal ('0X)	Volume	Value	Profit	Cash
Time to market	Normal project time from GT to GT5	<9 mo.	<6 mo.	X		X	X
Mid-range	Min BoM for The Corp phone	<$90	<$30	X		X	X
Platformisation Technology	# of Technology 66 Lic. shipping > 3M/yr	4	6	X		X	X
Interface	Interface units	>11M	>13M	X		X	X
Operator preference	Top-3 operators issue RFQ spec The Corp	1	2	X		X	X
Productivity							X
Get Torden	Lyn goes for Technology 66 in Sep-04	Yes		X		X	X
Fragmentation	Share of components modified	<10%	<5%		X	X	X
Commoditisation	Switching cost for a UI to another System	>1yr	>2yrs		X	X	X
Duplication	The Corp share of 'in scope' code in best-selling device	>90%	>95%		X	X	X
Competitiveness	Major feature comparison with MX	Same	Better	X		X	X
User experience	Key use cases superior vs. competition	5	10	X	X	X	X
Downstream cost saving	Project ROI for Licensees	>33%	>66%	X	X	X	X
Platformisation IFace	Number of shipping Lic.	33	55	X		X	X
Japan	Share of of XXXX sales	>50%	>60%	X		X	X
	Numbers are intentionally changed from real ones						

Business Values Quantified

3. Values are supported by Value Architecture

- Value Architecture: defined as:
  - anything you *implement* with a view to satisfying stakeholder values.
- Value Architecture:
  - includes product/system objectives
    - Which are a ‘design’ for satisfying stakeholder values
  - Has a multitude of performance and cost impacts
  - can impact a given system differently, depending on what is in the system, or what gets put in later
  - Needs to try to maximize value delivered for resources used.



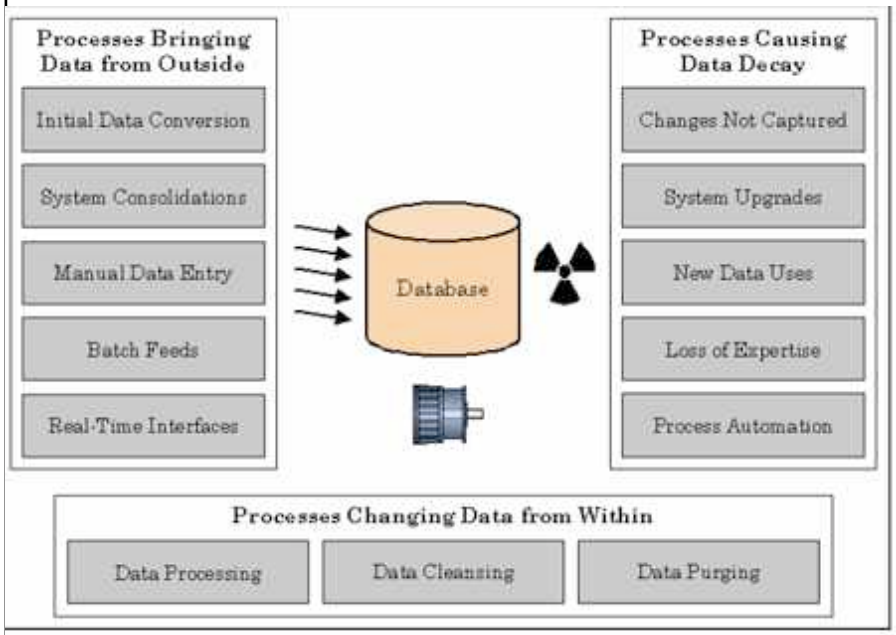
4. Value *levels* are determined by *timing*, *architecture effect*, and *resources*

Value levels: defined as:

the degree of satisfaction of value needs.

Value level:

- depends on *when* you observe the level
  - The environment, the people, other system performance characteristics (security, speed, usability)
- depends on the *current incremental power* of *particular value architecture* components
- depends on *resources available* both in development and operation

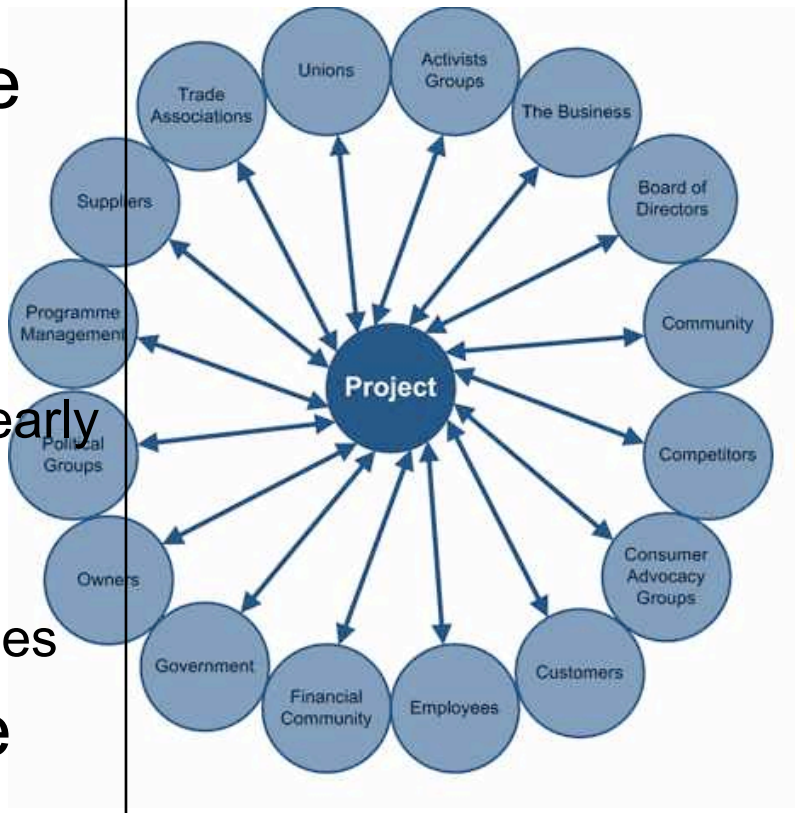


5. Required Value *levels* can differ for different scopes (where, who)

The level of value needed, and the level of value delivered - for a single attribute dimension (like Ease of Use) can vary for:

- different stakeholders
- at different times
  - (peak, holiday, slack, emergency, early implementation)
- for different 'locations'
  - countries, companies, industries

There is nothing simple like 'one level for all'



• 6. Value can be delivered *early*

You do not have to wait until ‘the project is done’ to deliver useful stakeholder value satisfaction.

You can intentionally target the highest priority stakeholders, and their highest priority value area, and levels.

You can deliver them early and continuously

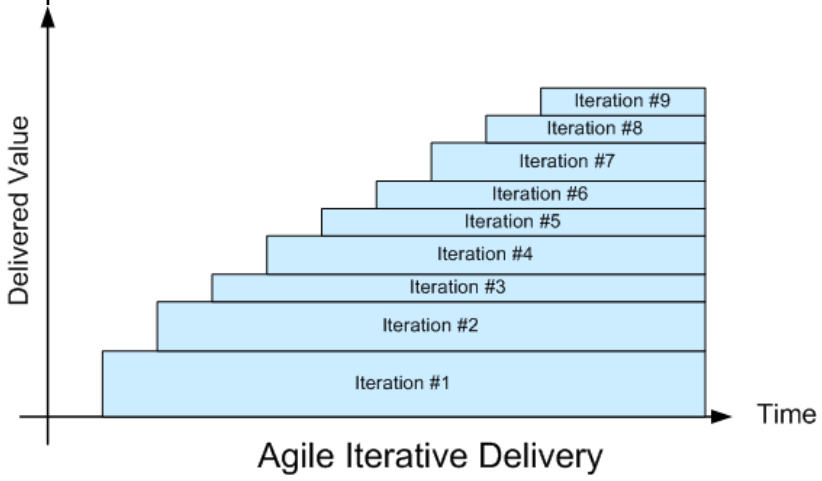
You can learn what is possible

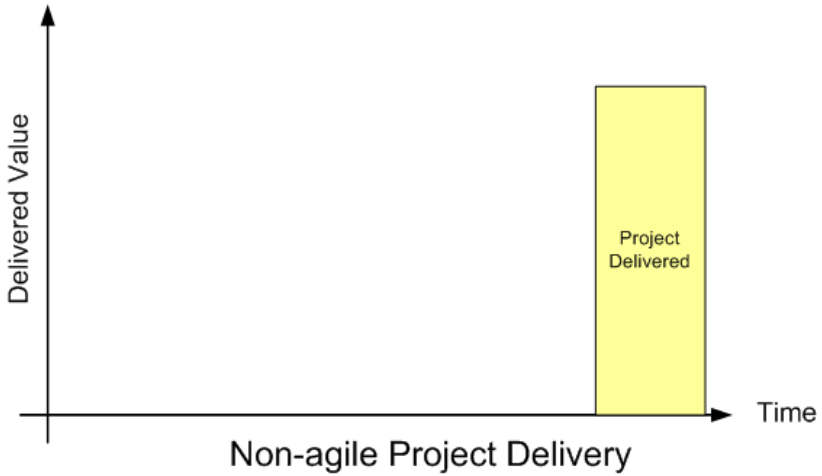
And what stakeholders really value.

Discover new value ideas

Discover new stakeholders

Discover new levels of satisfaction



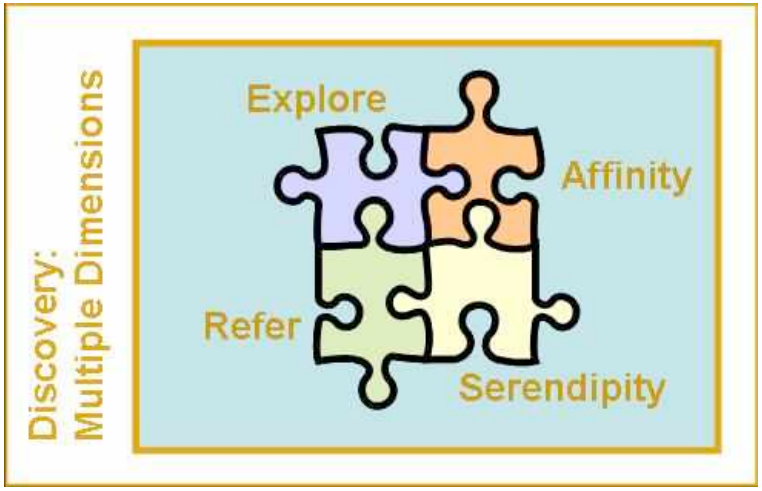


• 7. Value can be locked in incrementally

- You can increment the value satisfaction
  - *towards* longer term Goal levels
- You can spread the value deliveries
  - that are *proven* in *some* places,
  - more widely in the next increments
- This probably assumes that you have really handed over real results to real people.
  - Not just developed systems without delivery

8. New Values can be discovered (external news, experience)

- ! *Expect*, and try to discover,
  - –!entirely new stakeholder values.
- ! These will of course emerge *after you start delivering* some satisfaction, because:
  - –! Stakeholders believe you can help
  - –!Things *change*



9. Values can be *evaluated* as a function of *architecture* (using 'Impact Estimation')

- It is possible to get an **overview** of
  - the totality of impacts
  - that your **architecture**
  - (all designs and strategies)
  - **might** have
  - on all your defined stakeholder **needs**.

- Use an Impact Estimation table
  - and you will be able to spot *opportunities* for
    - high value and
    - low cost early deliveries
      - by analyzing the numbers on the table

		Viking Deliverables											
Business Objective	Weight	hardware adaptation	Telephony	Reference designs	IFace	Modularity	Defend vs Technology 66	Tools	User Experice	GUI & Graphics	Security	Defend vs OCD	Enterprise
Time to market	20%	20%	10%	30%	5%	10%	5%	15%	0%	0%	0%	5%	5%
Mid-range	10%	15%	0%	15%	0%	30%	15%	5%	10%	5%	5%	0%	0%
Platformisation Technology	5%	25%	10%	30%	0%	0%	10%	0%	5%	0%	10%	0%	5%
Interface	5%	5%	15%	15%	0%	5%	0%	5%	0%	0%	10%	0%	10%
Operator preference	10%	0%	10%	0%	15%	5%	20%	5%	10%	10%	20%	5%	10%
Get Torden	10%	25%	10%	10%	-10%	0%	20%	0%	10%	-20%	10%	10%	5%
Commoditisation	5%	20%	10%	20%	10%	-20%	25%	15%	0%	0%	5%	10%	5%
Duplication	10%	15%	10%	10%	0%	0%	40%	0%	0%	0%	5%	20%	5%
Competitiveness	5%	10%	15%	20%	0%	10%	20%	10%	10%	20%	10%	10%	10%
User experience	5%	5%	0%	0%	0%	20%	0%	0%	30%	10%	0%	0%	0%
Downstream cost saving	5%	15%	5%	20%	0%	10%	20%	0%	10%	0%	0%	10%	5%
Platformisation IFace	5%	10%	10%	20%	40%	0%	20%	5%	0%	0%	0%	0%	5%
Japan	5%	10%	5%	20%	0%	10%	0%	0%	10%	5%	0%	0%	0%
Contribution to overall result		15%	9%	17%	4%	7%	15%	6%	6%	1%	6%	6%	5%
Cost (£M)		£ 2.85	£ 0.49	£ 3.21	£ 2.54	£ 1.92	£ 2.31	£ 0.81	£ 1.21	£ 2.68	£ 0.79	£ 0.62	£ 0.60
ROI Index (100=average)		106	356	109	33	78	137	148	107	10	152	202	174

See next slide!
For enlargement!

Strategy Impact Estimation:
for a $100,000,000 Organizational Improvement Investment

Objectives

Technical Strategies

⇩ Defined! In earlier slide!

Strategy Impacts on Objectives

"Benefits"

Business Objective	Viking Deliverables: hardware adaptation	Telephony	Reference designs	IFace	Modularity	Defend vs Technology 66	Tools	User Experience	GUI & Graphics	Security	Defend vs OCD	Enterprise
Time to market	20%	10%	30%	5%	10%	5%	15%	0%	0%	0%	5%	5%
Mid-range	15%	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	5%	10%	5%	5%	0%	0%
Platformisation Technology	25%	10%	30%	0%	[illegible]	10%	0%	5%	0%	10%	0%	5%
Interface	5%	15%	15%	0%	5%	0%	5%	0%	0%	10%	0%	10%
Operator preference	0%	[illegible]	[illegible]	[illegible]	[illegible]	20%	5%	10%	10%	20%	5%	10%
Get Torden	25%	10%	10%	-10%	0%	20%	0%	10%	-20%	10%	10%	5%
Commoditisation	20%	10%	20%	10%	-20%	25%	15%	0%	0%	5%	10%	5%
Duplication	15%	0%	10%	0%	0%	40%	0%	0%	0%	5%	20%	5%
Competitiveness	10%	15%	20%	0%	10%	20%	10%	10%	20%	10%	10%	10%
User experience	5%	[illegible]	0%	0%	[illegible]	0%	0%	30%	10%	0%	0%	0%
Downstream cost saving	15%	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	10%	0%	0%	10%	5%
Platformisation IFace	10%	10%	20%	40%	0%	20%	5%	0%	0%	0%	0%	5%
Japan	10%	5%	20%	0%	10%	0%	0%	10%	5%	0%	0%	0%
Contribution to overall result	15%	9%	17%	4%	7%	15%	6%	6%	1%	6%	6%	5%
Cost (£M)	£ 2.85	£ 0.49	£ 3.21	£ 2.54	£ 1.92	£ 2.31	£ 0.81	£ 1.21	£ 2.68	£ 0.79	£ 0.62	£ 0.60
ROI Index (100=average)	106	358	109	33	78	137	148	107	10	152	202	174

Cost

358 !

10. Value delivery will attract resources.

- If you are really good at delivering value
  - You can expect to attract
    - even more funding
  - Managers like
    - to be credited with success
  - Money seeks
    - best interest rates

Gilb's Value Manifesto: A Management Policy?

1.! Really useful value, for real stakeholders will be defined measurably.
No nice-sounding emotive words please.

2.! Value will be seen in light of total long term costs as a decent return on investment.

3.! Powerful management devices, like motivation and follow-up, will make sure that the value for money is really delivered –
or that the failure is punished, and the success is rewarded.

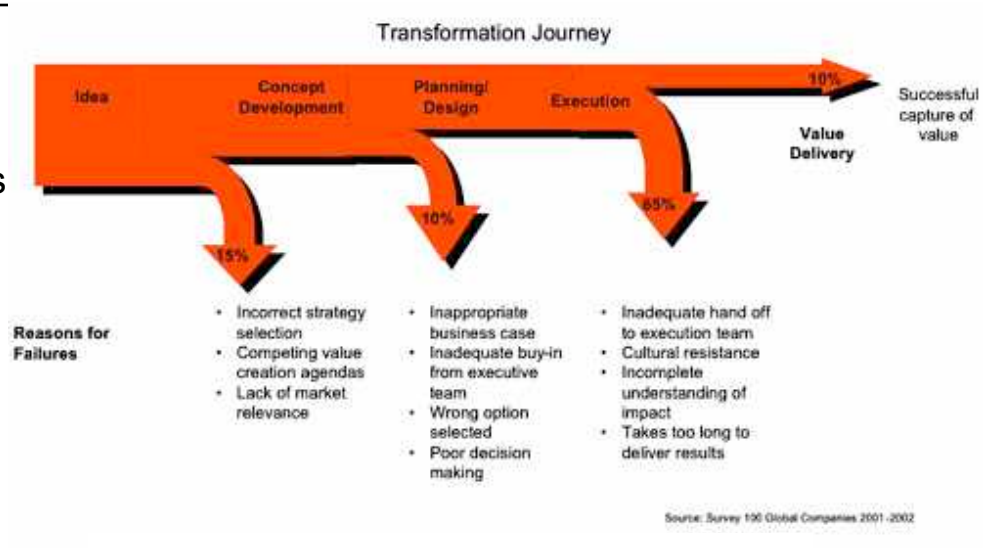
4.! The value will be delivered evolutionarily –
not all at the end.

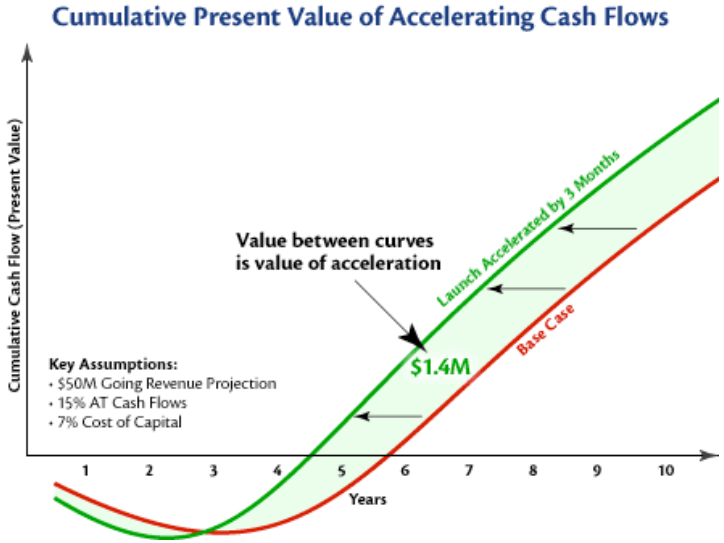
5.! That is, we will create a stream of prioritized value delivery to stakeholders, at the *beginning* of our value delivery projects;
and continue as long as the real return on investment is suitably large.

6.! The CEO is primarily responsible for making all this happen effectively.

1.! The CFO will be charged with tracking all value to cost progress.

2.! The CTO and CIO will be charged with formulating all their efforts in terms of measurable value for resources.





Source "Value Delivery in Systems Engineering" available at www.gilb.com
Unpublished paper http://www.gilb.com/community/tiki-download_file.php?fileId=137

The Value Delivery Problem

- Sponsors who order and pay for systems engineering projects, must justify their money spent based on the expected consequential effects (hereafter called ‘value’) of the systems.
-
- The value of the technical system is often expressed in presentation slides and requirements documents as a set of nice-sounding words, under various titles such as “System Objectives”, and “Business Problem Definition”

Some Assertions

Assertion 1. When top management allows large projects to proceed, with such badly formulated primary objectives, then

- –! they are responsible as managers for the outcome (failure).
- –! They cannot plead ignorance.

Assertion 2. The failure of technical staff (project management) to react to the lack of primary objective formulation by top management is also a total failure to do reasonable systems engineering.

- –! Management might have a poor requirements culture, but we should routinely save them from themselves.

Assertion 3. Both top managers and project personnel can be trained and motivated to clarify and quantify critical objectives routinely.

- –! But until the poor external culture of education and practice changes, it may take strong CEO action to make this happen in your corporation.
- –! My experience is that no one else will fight for this.

Assertion 4. All top level system performance improvements, are by definition, variables.

- –! So, we can expect to define them quantitatively.
- –! We can also expect to be able to measure or test the current level of performance.
- –! Words like ‘enhanced’, ‘reduced’, ‘improved’ are not serious systems engineering requirements terms.

For example:
(Real, engineering system, but doctored for anonymity)

1. *Central to The Corporations business strategy is to be the world's **premier** integrated_ <domain> service **provider**.*

2. *Will provide a much more efficient **user** experience*

3. *Dramatically scale back the **time** frequently needed after the last data is acquired to time align, depth correct, splice, merge, recompute and/or do whatever else is needed to **generate** the desired **products***

4. *Make the system much **easier** to **understand** and **use** than has been the case for previous system.*

5. *A primary goal is to provide a much more **productive** system **development** environment than was previously the case.*

6. *Will provide a richer set of functionality for **supporting** next-generation logging **tools** and applications.*

7. ***Robustness** is an essential system requirement (see rewrite in example below)*

8. *Major improvements in **data quality** over current practices*

For Example:

I rewrote the top level system requirement in the above example using Planguage [Gilb 2005]:

"7. Robustness is an essential system requirement."

to be:

Rock Solid Robustness:

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

•!

Software Downtime:

Type: Software Quality Requirement. **Version**: 25 October 2007.

Part of: Rock Solid Robustness.

Ambition: to have minimal downtime due to software failures <- HFA 6.1

Issue: does this not imply that there is a system wide downtime requirement?

Scale: <mean time between forced restarts for defined [Activity], for a defined [Intensity].>

Fail [Any Release or Evo Step, Activity = Recompute, Intensity = Peak Level] 14 days <- HFA 6.1.1

Goal [By 2008?, Activity = Data Acquisition, Intensity = Lowest level] : 300 days ??

Stretch: 600 days.

Restore Speed:

Type: Software Quality Requirement. **Version**: 25 October 2007.

Part of: Rock Solid Robustness

Ambition: Should an error occur (or the user otherwise desire to do so), the system shall be able to restore the system to a previously saved state in less than 10 minutes. <-6.1.2 HFA.

Scale: Duration from Initiation of Restore to Complete and verified state of a defined [Previous: Default = Immediately Previous]] saved state.

Initiation: defined as {Operator Initiation, System Initiation, ?}. Default = Any.

Goal [Initial and all subsequent released and Evo steps] 1 minute?

Fail [Initial and all subsequent released and Evo steps] 10 minutes. <- 6.1.2 HFA

Catastrophe: 100 minutes.

Testability:

Type: Software Quality Requirement.
Part of: Rock Solid Robustness
Initial Version: 20 Oct 2006
Version: 25 October 2007.
Status: Demo draft,
Stakeholder: {Operator, Tester}.
Ambition: Rapid-duration automatic testing of <critical complex tests>, with extreme operator setup and initiation.

Scale: the duration of a defined [Volume] of testing, or a defined [Type], by a defined [Skill Level] of system operator, under defined [Operating Conditions].

Goal [All Customer Use, Volume = 1,000,000 data items, Type = WireXXXX Vs DXX, Skill = First Time Novice, Operating Conditions = Field, {Sea Or Desert}. <10 mins.

Design Hypothesis: *Tool Simulators, Reverse Cracking Tool, Generation of simulated telemetry frames entirely in software, Application specific sophistication, for drilling – recorded mode simulation by playing back the dump file, Application test harness console <-6.2.1 HFA*

the problem with conventional requirements

- their source or authority
  - may be *undocumented* and *unknown*
- they are probably not at all clear
  - about *exactly* what should happen,
  - where or when, or under which conditions
- there is no contract,
  - to pay *only* upon such *results* being *delivered*
- there is no specific design or architecture,
  - to *enable* the technical product to achieve the requirements

£50 million Wasted

- The above example was the basis in 1999 for a project that had
  - in 2006 spent over $100 million,
  - for 8 years
  - and had never delivered any value whatsoever to the corporation.
- There was never any quantified or testable definition of the requirements.
- There was never any direct link
  - from the project activity, requirements, or architecture,
  - to these primary top management
    - (CEO and next level directors) objectives.
- The project was doomed from the start.

Another Real (Doctored) Example: Financial Corp. Top Level Project requirements

1. Reduce the costs associated with managing redundant / regionally disparate systems.
2. Single global portfolio management system.
3. Reduce overall spending with a reduction in redundant initiatives.
4. Governance structures - system agnostic.
5. All projects in project portfolio system.
6. Reduce development project spend on low priority work with better alignment between Technology and business demand.
7. Project portfolio Framework, Business Value metrics for prioritization.
8. Reduction in cost over runs.
9. Definition criteria for project success.
10. Metrics and exception reporting for cost management.
11. Linkage of actual costs to forecast.
12. Increase revenue with a faster time to market.
13. Knowledge management, project ramp up templates.

The Financial System

- This project spent about $50 million, in a single year.
- Responsible management, impatient for some results, discovered to their horror, through an audit, that the above primary objectives had **never been clarified or taken seriously.**
- The responsible ('former') project manager had chosen to **ignore** the opportunity, planned by a major component supplier, to **clarify** these objectives.
- The project manager spent a lot of effort obtaining 'requirements from users',
  - but no further effort on *these* **primary** objectives above.
- Serious effort was, after the audit, then immediately spent quantifying and taking seriously these primary objectives.
- It took a single day to draft a quantified version.
- The quantified version made a clear distinction between
  - technical objectives (system quality – examples 2 and 5 above) and
  - stakeholder values (making the business better, examples 8 and 12 above).

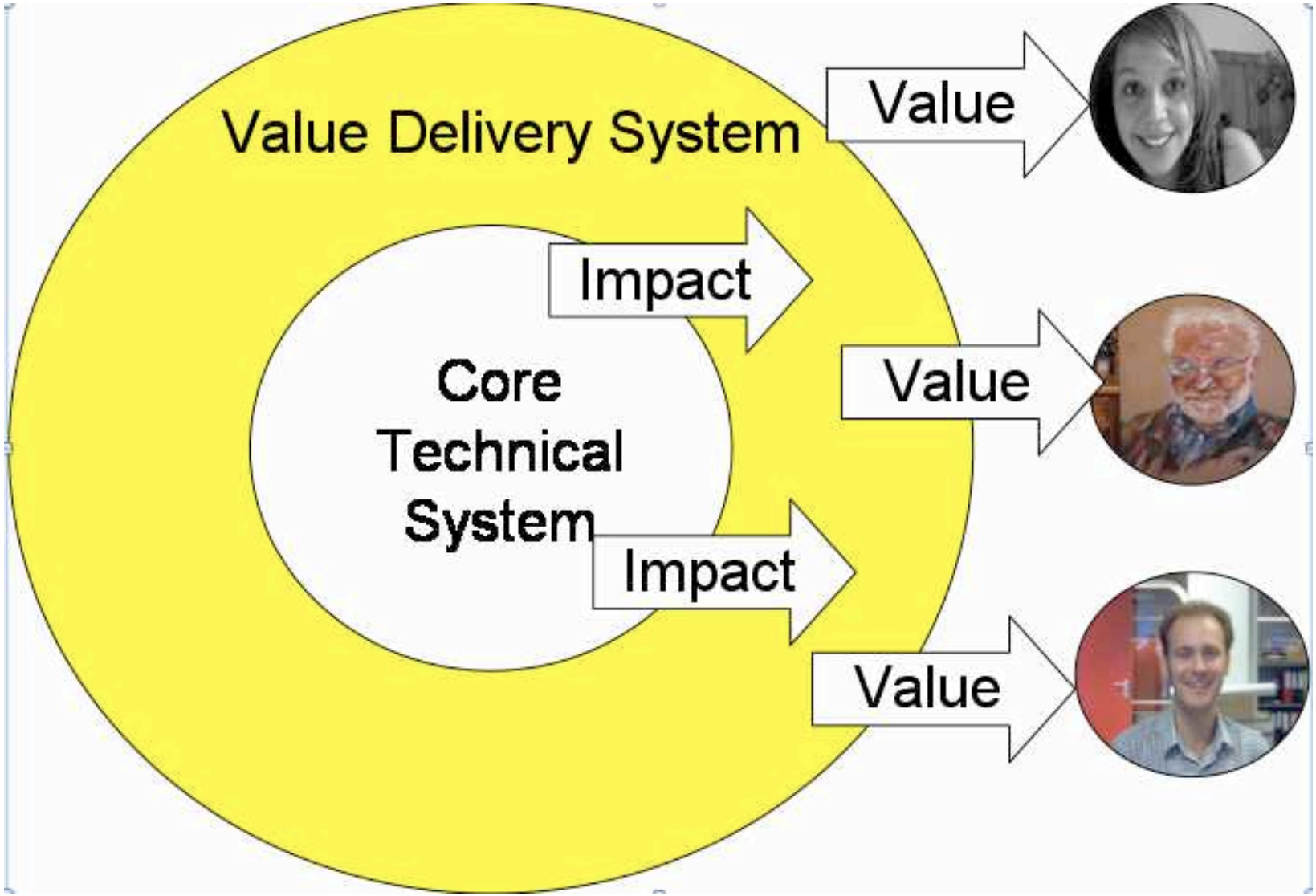
Another Assertion Delivering Value

- Assertion 5.
  - If the hardware/software systems supplier is
    - not prepared to deal with the system level that delivers the value from their product,
    - then someone,
      - internally or an external contractor
    - needs to undertake the project of delivering the value expected.

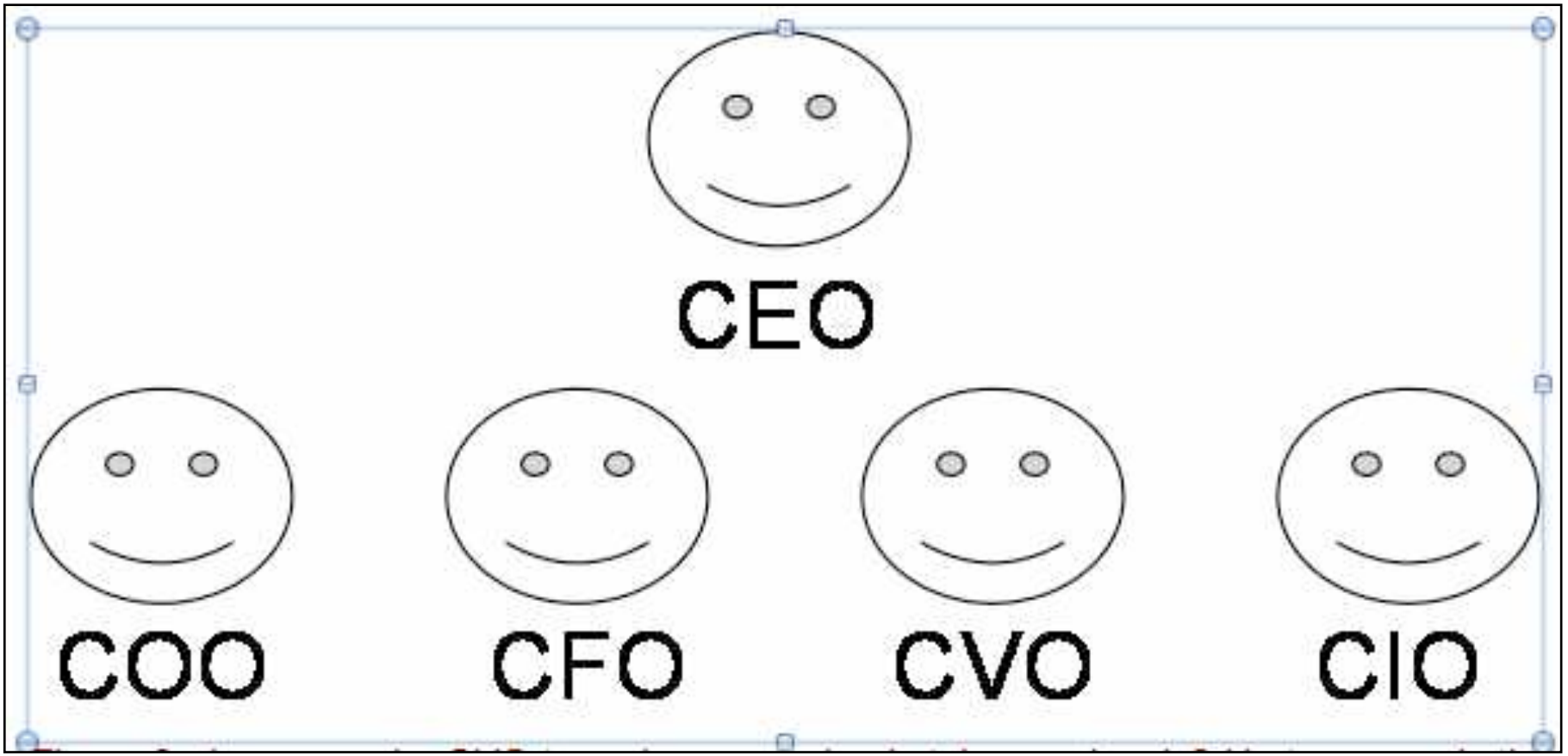
Assertion 6.
Systems Engineering for Value

- This 'value delivery process' is
  - likely to entail considerable human and organizational aspects,
  - and little hardware and software technology.
- So it may be inappropriate work for systems engineers
  - who are not expert in, and committed to, the social, political, and organizational aspects of systems engineering.
- But of course this 'social' ability
  - is a necessary and valid component of full systems engineering –
  - or we cannot call it 'systems' engineering
  - and exclude the social, political system aspects.
-

Value delivery is NOT Technical Construction



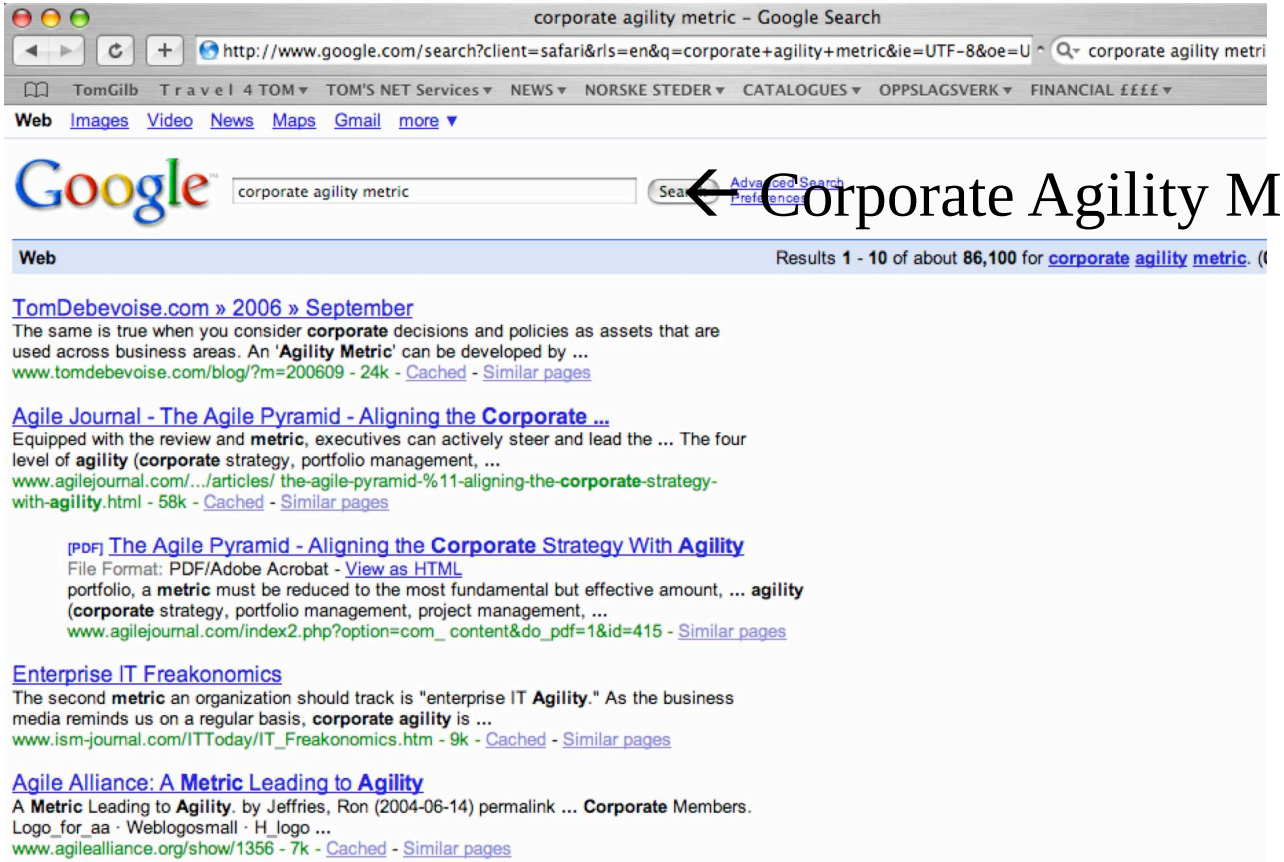
Do we need a Chief Value Officer?



The Value Principles:

1. Value can always be articulated quantitatively, so that we can understand it, agree to it, track it, contract for it and understand it in relation to costs.
2. Value is a result, delivered to a real set of stakeholders.
3. Value must be seen in light of lifetime total cost aspects, and must be as profitable as alternative investments.
4. Value occurs through time, as a stakeholder experience: it is not delivered when a system to enable it is delivered – only when that system is successfully used to extract the value.
5. Value can be delivered early, and for part of one stakeholder's domain. This proves the value potential, and actually improves the real organization.
6. There is never a really sufficient reason to put off value delivery until large-scale long-term investments are made. This is just a common excuse from the many weak, ignorant, cowards who would like to spend a lot of money before being held to account.
7. People who cannot deliver a little value early, in practice, cannot be entrusted to deliver a lot of value for a larger investment.
8. The top management must be primarily responsible for making value delivery happen in their organization. The specialist managers will never in practice take the responsibility, unless they are aiming to take over the top job.
9. Value is a multiplicity of improvements, and certainly not all related to money or savings – but we still need to quantify the value proposition in order to understand it, and manage it.
10. If we prioritize highest value for money first, then we should normally experience an immediate and continuous flow of dramatic results, that the entire organization can value and

1. Value can always be articulated quantitatively, so that we can understand it, agree to it, track it, contract for it and understand it in relation to costs.



← Corporate Agility Metric!

•! If all else fails, Google it!

2. Value is a result delivered to a real set of stakeholders.

- Value is not ‘activated’ by a technical performance characteristic alone,
  - like Usability, security or Robustness.
- It is only created when it meets real people in their everyday stakeholder situation of work:
  - Call Center, Battlefield Analyst, Corporate Trader.
- It has to save them time, or make their work better.
- The value created by the interaction with a stakeholder type may be cumulated every time the system is used for some new activity, customer, transaction, or decision.
- It may be cumulated by a very large number of that type of stakeholder (10,000 sales people). And through a very long time (years).
- It is obvious from this common sense observation that value is *not* created by the technical system performance characteristics (speedy response, user friendly),
  - but by making those technical system characteristics available
    - in practice
    - to as many real people, and
    - as many transactions, and
    - for as long a time as possible.

3. Value must be seen in light of lifetime total cost aspects, and must be as profitable as alternative investments.

- We cannot allow ourselves to be blinded narrowly by quantified value.
- We must constantly estimate, and manage the value for money: the return on investment.
- And if the costs of delivering the value get out of hand, and exceed the value –
  - it is time to either reengineer the system
  - or decommission it.
  - Who will do this if not some constant CVO vigilance?

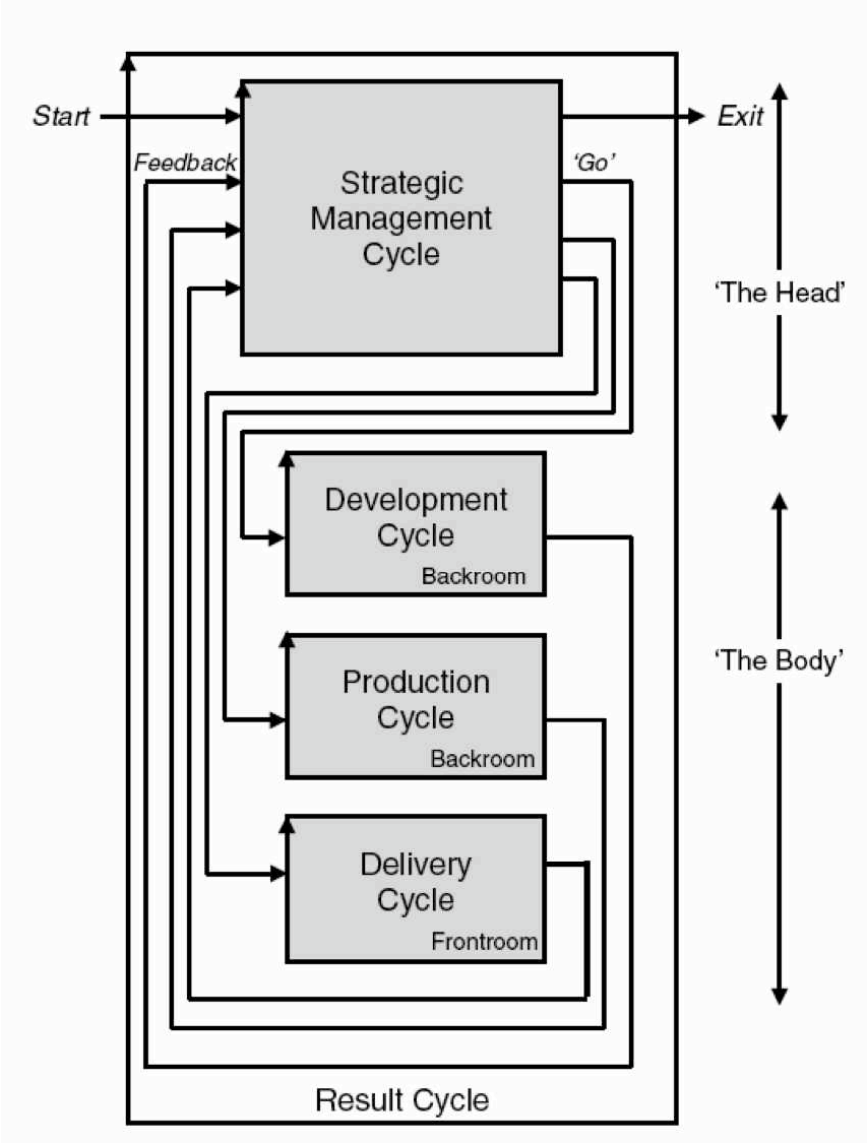
4. Value occurs through time, as a stakeholder experience: it is not delivered when 'a system to enable it' is delivered – only when that system is successfully *used* to extract the value.

- A **conscious strategy**, and **conscious formal plan**, must be made to deploy a technical system so that the value is delivered.
- We have to deal with political problems – like power centers (trade unions, management fiefdoms) and economic waste centers.
- We have to motivate people to give up their comfortable older systems and deploy scary new ones.
- We have to support the correct use by
  - training, call centers, local consultancy, measurement and feedback on the technical system,
  - is it actually delivering what we need, in order to get people to use it at all, to use it well?
- feedback on the stakeholder environments it is deployed in:
  - are they happy with it?
  - Do they have improvement suggestions?
  - Are there undesired variations in costs and benefits?
- feedback on deployment to the entire scope of stakeholders,
  - in relation to time plans:
  - is it being deployed successfully rapidly enough?
-
- Obviously this should be the natural concern and use of true systems engineering.
  - But in fact, there is little in the training, the conferences, the handbooks [INCOSE SE Handbook], to verify that systems engineering as a discipline has matured to the point where these concerns are safely included.
  - We are still too much 'engineers' (techies); and know and care too little about value management, and the organizational and management culture part of our domain.

5. Value can be delivered early, and for *part* of one stakeholder's domain. This proves the value potential, and actually improves the real organization.

- Our systems development culture is still very much a 'waterfall' culture.
- Finish the big system, and then deploy it [INCOSE SE Handbook 2-3, and 3-2 for example].
- There was no visible mention, in the Handbook, of a true evolutionary life cycle (even though the US DoD adopted one for software at least long ago, DoD Mil Std 498).
- There is no notion of early, frequent and gradual delivery of results to stakeholders, even though that has been practiced successfully in many large military, space and software systems for decades [Larman].
- Big Bang is still our mentality.
- I helped Douglas/Boeing to do value delivery Evolutionary projects for 25 aircraft projects in 1990. It was an unknown concept for them, but it was easily doable by every team we did it on; in real projects. We use 'next week' as our measure of when we would produce some useful value.
- I know that this sounds incredible and impossible to conventional ears. But it is simple enough in practice, and very close indeed to weaponry progress during the Second World War [Discovery Channel!].

Intelligent Feedback About Value



6. There is never a really sufficient reason to put off value delivery until large-scale long-term investments are made.

This is just a common excuse from those who would like to spend a lot of money before being held to account.

- There are vested interests who will happily consume public and private corporate money forever and deliver failure or little or no real value.
- The consumer and their representatives seem happy to contract for *effort*, but not contract for *value*.
- I cannot believe there are so many foolish people with so much money as I have had occasion to observe in practice
  - (example the $50 to $100 million wasted projects at the beginning of this paper, which are in fact small by comparision with some; like documented DoD waste in software engineering alone ($20 billion annually, many years ago).
- This is not necessary! We could avoid it by contracting for value and results. [Gilb, No Cure No Pay]. This is hardly on the agenda, and not discussed at all in the INCOSE Handbook.
- It would require two technical pieces of knowledge
  - The ability to quantify and measure value
  - The ability to decompose large projects into much smaller increments of value delivery.
- These exist, but the 'will to contract for value' does not.
- Some management leadership please!

7. People who cannot deliver a little value early in practice, cannot be entrusted to deliver a lot of value for a larger investment.

- Ericsson of Sweden, who learned to deliver mobile telephone base stations in 1990 in monthly evolutionary steps observed this principle (Jack Järkvik).
- If you are going to spend $100,000,000 before anything happens, and nothing then does.
  - It might have been a good idea to offer the project or supplier a mere $1 million (1%)
    - and ask if they could create some of the long-term projected value for that 1% of budget.
    - If they cannot, then there is no reason to believe they will use your $100 million wisely.
    - If they can; do so, then feed them millions, one at a time until it is no longer profitable!

8. The top management must be primarily responsible for making value delivery happen in their organization. The specialist managers will never, in practice, take the responsibility, unless they are aiming to take over the top job.

- Top management, the CEO, needs to decide they are primarily responsible for value for money, and dictate a policy of focus on 'value for money' (see earlier in this paper for policy ideas).
- One excellent CEO client of mine who did so, Robb Wilmott of ICL UK (23,000 employees then), turned years of losses into 14 straight years of profit for his computer company – unlike competitors, like IBM, at the time. My observation was:
- • it only happened because the CEO threatened all other top managers with loss of power and budget if they did *not* 'quantify the value' they were going to deliver
- • they began to think clearly about their responsibilities, perhaps for the first time
- • it helps if the CEO is an engineer, not an MBA ☺
- Another UK CEO, pulled the same trick – about 2003.
  - But had to fire the marketing director, and the sales director, for refusing to really play ball.
  - Some directors have a real fear of being specific about what they are responsible for.
  - Interestingly the current Chairman of *this* company was one of the above-mentioned ICL Directors (Marketing) who we trained to quantify, things like the primary new product line vision, 'Adaptability' of his product.

9. ‘Value’ is a multiplicity of improvements, and certainly not all related to money or savings – but we still need to quantify the value proposition in order to understand it, and manage it.

- I strongly dislike value schemes that try to turn all values into money. Do they really think management understands no other concept?
-
- Peter Drucker, I think it was (Management By Objectives, in ‘The Practice of Management’), established long ago that no corporation is driven by money alone. Thus the Balanced Scorecard, to retain some non-financial balance, I suppose.
- If the value you are aiming at is for example, ‘increased potential customer willingness to shortlist you’,
  - then there is an estimable money value for that,
  - but I would be afraid of losing focus on the short-listing, by converting this idea to money.
- You would need to measure the quantity of real short-listing to manage that value, for example.
  - I believe you need to state and measure things directly,
  - especially of you want to track early lead indicators of value –
  - and keep people focused on a dynamic and changing situation.

10. If we prioritize highest value for money first, then we should normally experience an immediate and continuous flow of dramatic results, that the entire organization can value and relate to. Be deeply suspicious of long-term visions with no short-term proof.

- We should try to skim the cream off the top.
  - With early realistic feedback, and changing technology and markets, we should be able to avoid a dramatic diminishing return on investment for some time.
- Projects, at one extreme, should be practically self-funding;
  - or at least not in need of huge initial budgets, then overspent by factor 3.14 (Pie instead of 'piece of cake') before management feels uncomfortable
- You have a lot of choice, in spite of some dependencies,
  - to 'cherry pick' very high value for money, early deliveries.
  - Not exactly a new marketing technique –
    - but maybe alien to our Defence Supplier Systems Engineering mentality.
- Again, if we contracted to pay them for value for money,
  - they would be more focussed on making it happen.
  - This is *our* problem, not theirs.
  - We fail to motivate suppliers to do the right thing for us.
- We fail to even discuss this in our systems engineering literature.
  - We have progress payments, but not based on value delivery, early and frequently.
  - 'Payment Schedules' (sounds nice and bureaucratic) are mentioned in the SE Handbook, but not 'Value Payments'.
  - We need to extend the concept!

Summary

- Top management needs to change their culture
  - to manage the actual delivery of real value,
  - and not leave it to systems engineers to drive this change.
- Systems Engineers can execute the value engineering and delivery –
  - but only top management can make it happen.