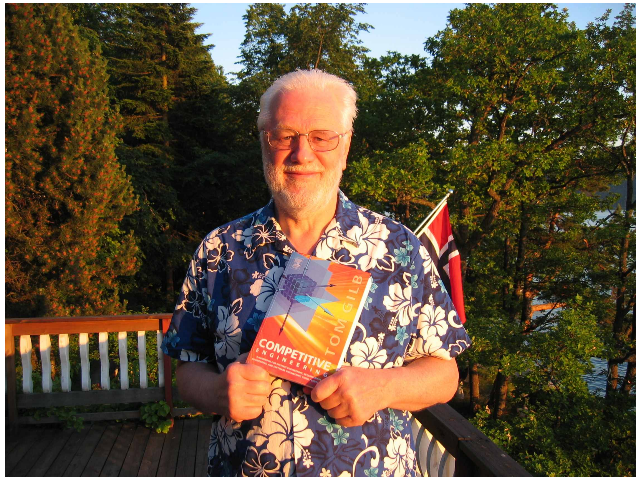
‘Lean’ Quality Assurance: beats conventional *testing*

(a 1 hour lecture (at breakneck speed!) based on a 1-2 day course ☺)



Presenter: Tom Gilb, Gilb.com

Oslo NDC, 16 June 2010, 15:00 to 16:00

tomsgilb@gmail.com

@imtomgilb

www.gilb.com

These slides will be on gilb.com downloads

Based on a paper from ‘Testing Experience’

http://www.gilb.com/tiki-download_file.php?fileId=288

Simple Summary

- ! If you think improving your testing is a smart way to get better software quality
  - !You are wrong
  - !Consider some of these more efficient options

Capers Jones

Defect *Removal* Effectiveness
Inspections and Tests

A REMINDER OF HOW BAD TEST & INSPECTION METHODS ARE

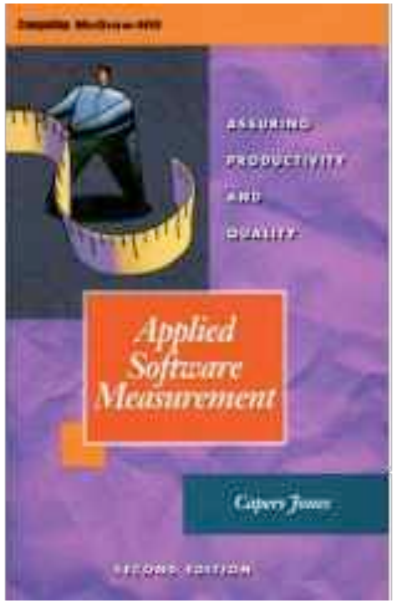

ASSURING PRODUCTIVITY AND QUALITY
Applied Software Measurement
Capers Jones
SECOND EDITION

Table 9: Software Defect Removal Effectiveness Ranges (Capers Jones)

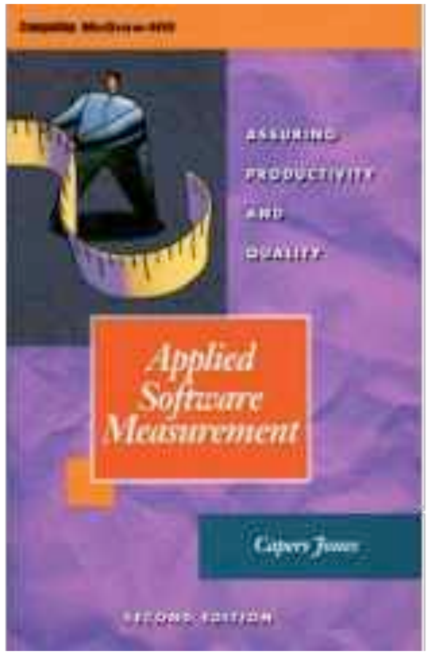
Defect Removal Activity	Ranges of Defect <u>Removal</u> Effectiveness
<u>Informal</u> design reviews	25% to 40%
<u>Formal design inspections</u>	45% to 65%
Informal code reviews	20% to 35%
<u>Formal code inspections</u>	45% to 70%
<u>Unit test</u>	15% to 50%
New function test	20% to 35%
Regression test	15% to 30%
Integration test	25% to 40%
Performance test	20% to 40%
<u>System test</u>	25% to 55%
Acceptance test (1 client)	25% to 35%
Low-volume Beta test (< 10 clients)	25% to 40%
High-volume Beta test (> 1000 clients)	60% to 85%

Capers Jones

No 'Silver Bullet' Solution Machine Guns Kill Defects

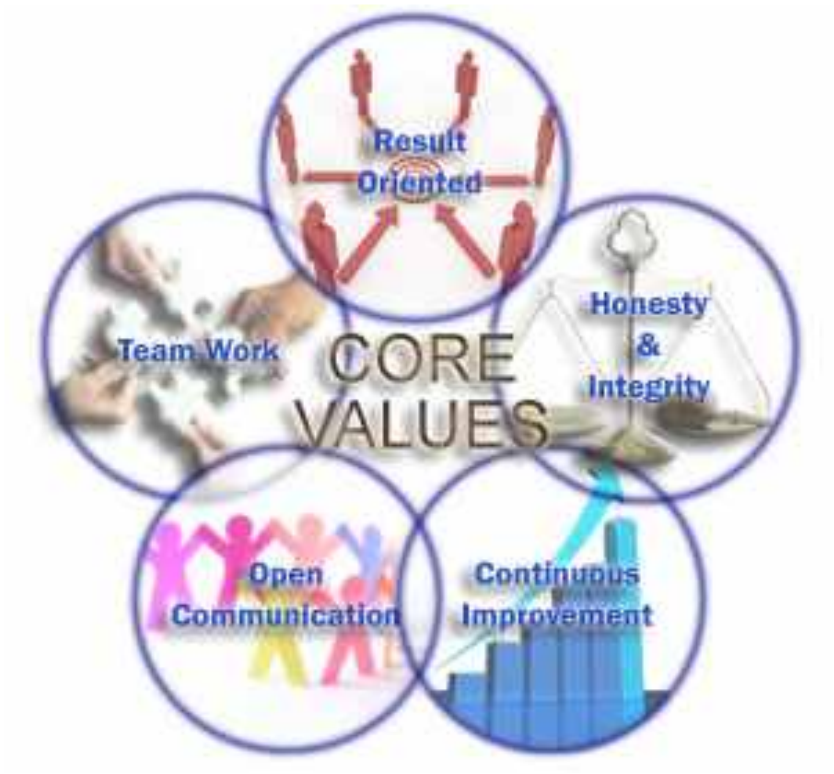
They give little hope of 'zero defects'

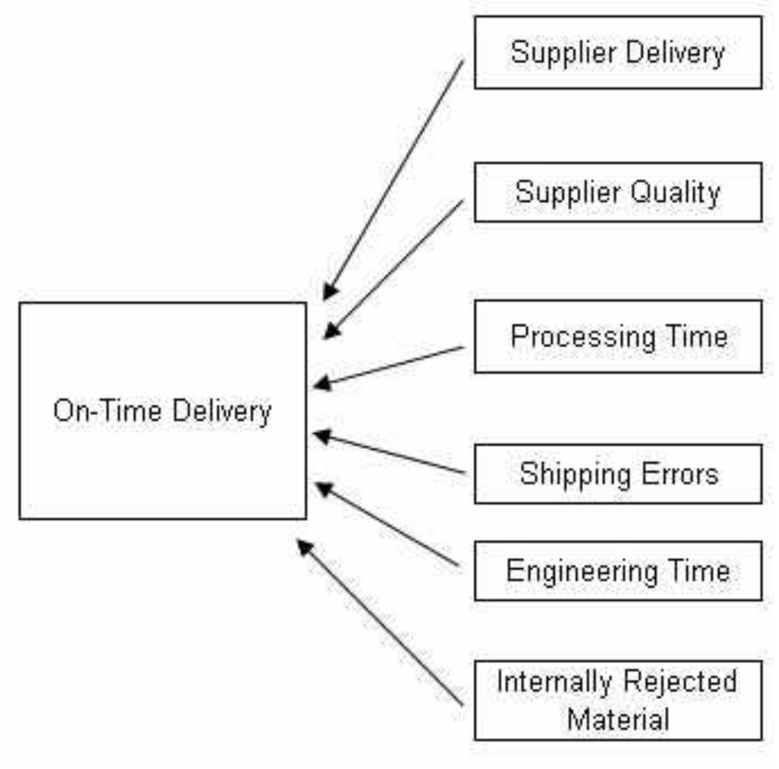
- "It is obvious that no single defect removal operation is adequate by itself.
- This explains why
  - "best in class" quality results can only be achieved from
    - synergistic combinations of
      - defect prevention,
      - reviews or
      - inspections,
      - and various kinds of test activities.
- **Between eight and 10 defect removal stages are normally required to achieve removal efficiency (he means 'effectiveness') levels > 95%".**
    - Jones, Capers; <u>Applied Software Measurement</u>; McGraw Hill, 2nd edition 1996; ISBN 0-07-032826-9; 618 pages.


ASSURING PRODUCTIVITY AND QUALITY
Applied Software Measurement
Capers Jones
SECOND EDITION

What is ‘Quality’ ?

- ! Can ‘Quality’ be measured?





What is the *smartest* way to get software system quality?

Some sub-questions

- **What is the role of Testing?**
- **What does ‘QA’ mean?**
- **How does it differ from ‘QC’?**
- **Does ‘Agile’ relate to quality?**
- **Does ‘Lean’ relate to quality?**
- **Is there one ‘best’ method?**
- **Does it ‘depend’?**

The QA Dilemma

- !Do you **want to test**.
  - !*Even if you do* <u>not</u> *get quality?*
- !Or do you **want** to get **quality**.
  - !*Even if you do* <u>not</u> *test?*

QA 1: Serve All Real Stakeholders:

- Many (30-40) multiple stakeholders to consider in QA:
- not just 'user' and 'customer'.
- This is a Scrum 'Product owner' responsibility:
  - but how well is it done in practice?
- We believe it is done ***badly,***
  - and have constructive advice for doing it better.

Stakeholder: Concept.!

‘Stakeholders’ are:

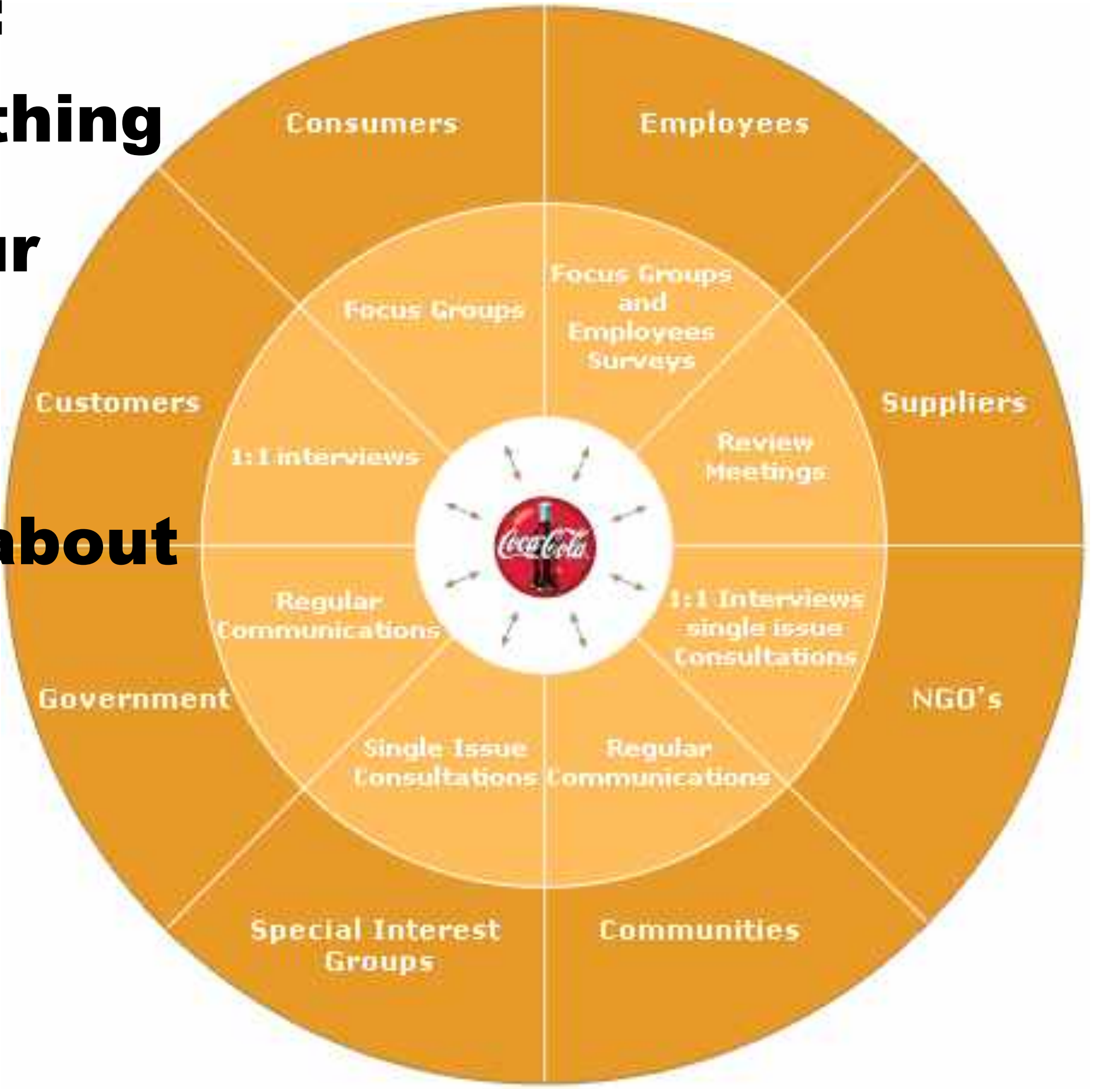
Any person, group or thing

that can determine our systems degree of success or failure,

by having an opinion about

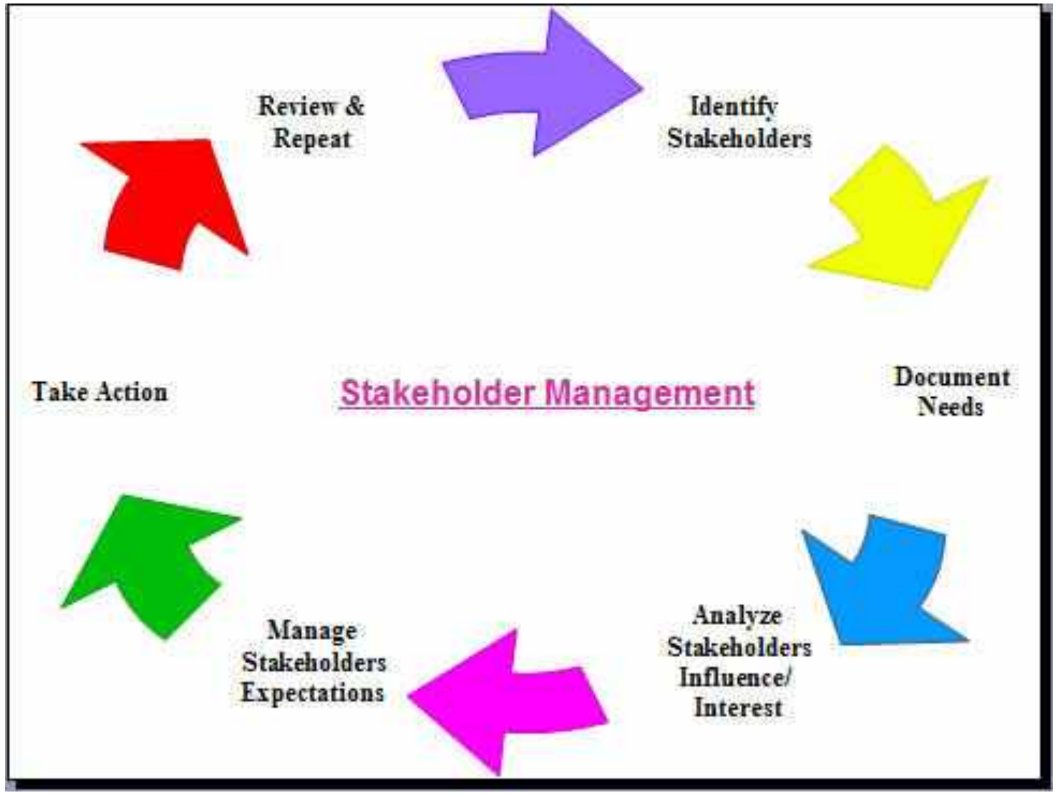
system performance characteristics and

system lifecycle constraints



Stakeholder Interests

1. *Setting the* ***objectives*** *for a process.*
2. ***Evaluating*** *the quality of the product*
3. ***Using*** *the product or system, even indirectly*
4. ***Avoiding*** *problems for themselves as a result of our product or system.*



Stakeholder Map

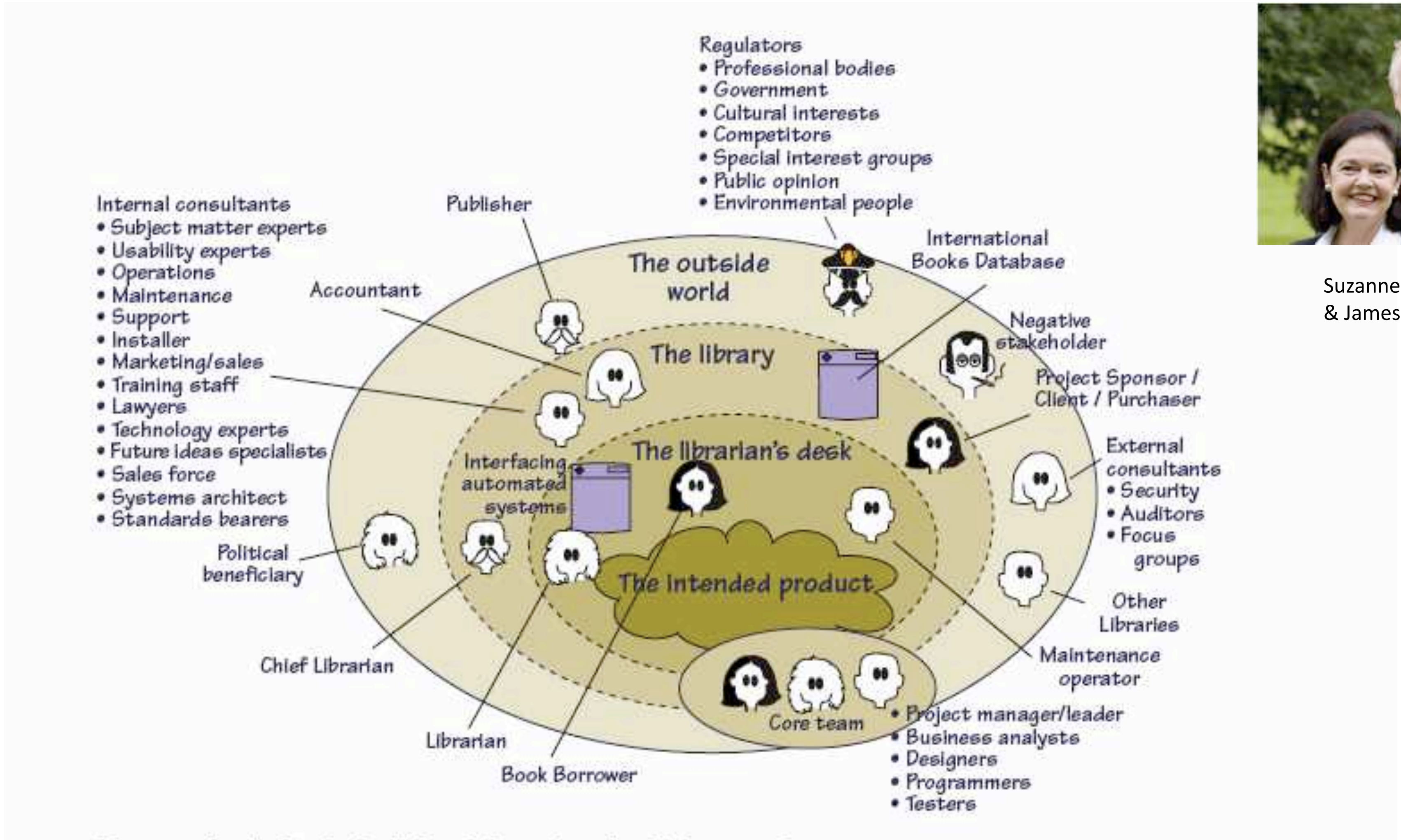


Figure 1: A Stakeholder Map for the Library Loans project

Suzanne Robertson
& James Robertson

Copyright The Atlantic Systems Guild, Used with Kind Permission.

http://www.requirementsnetwork.com/sites/requirementsnetwork.com/files/Volere_Requirements-A_Socio_Technical_Discipline.pdf

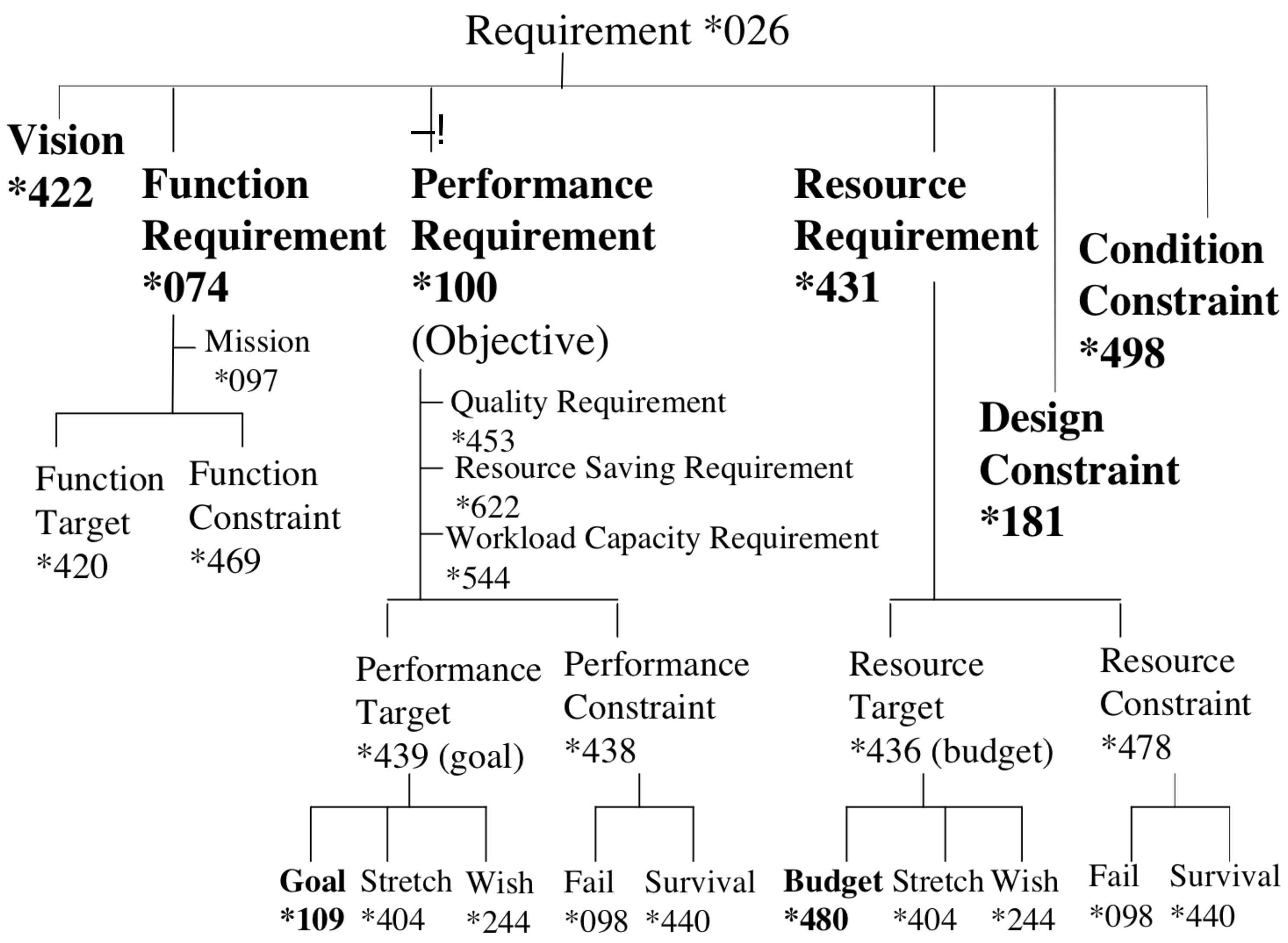
‘Requirement’ is a

•! A ‘requirement’ is a

–!“stakeholder valued future state”.

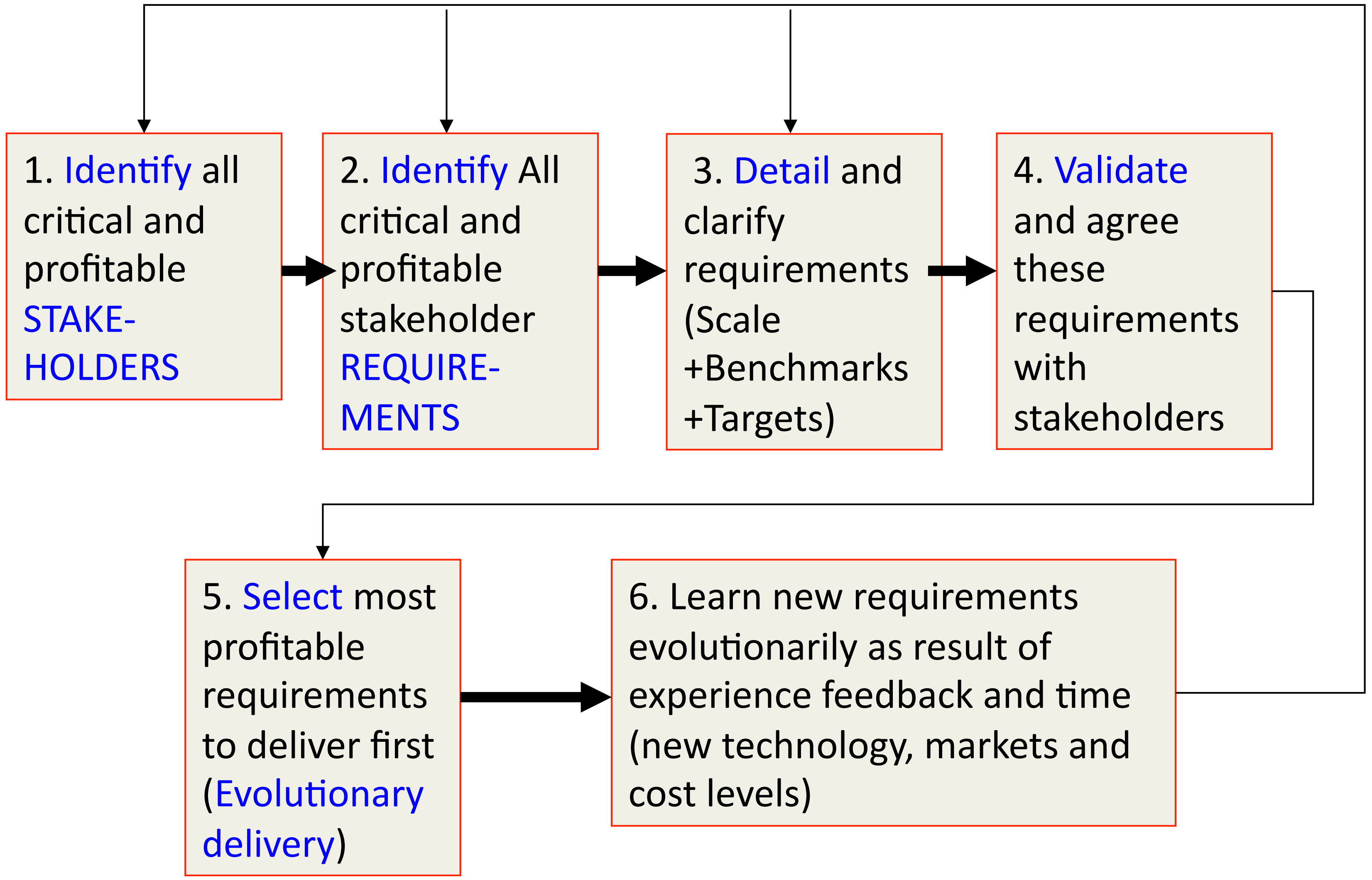
Fn
Past
Goal

•! –



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

Stakeholders:
How to find out about, and confirm, their requirements


1. Identify all critical and profitable STAKE-HOLDERS
2. Identify All critical and profitable stakeholder REQUIRE-MENTS
3. Detail and clarify requirements (Scale +Benchmarks +Targets)
4. Validate and agree these requirements with stakeholders
5. Select most profitable requirements to deliver first (Evolutionary delivery)
6. Learn new requirements evolutionarily as result of experience feedback and time (new technology, markets and cost levels)

Stakeholders: Quality

- In order to understand QUALITY
- You have to understand STAKEHOLDERS
  - And the qualities they prioritize

Practical Actions: Stakeholders

- Policy (Use to Audit Processes)
  - All projects will thoroughly analyze their critical stakeholders, the stakeholder needs, and their priorities.
- Rules (Standards for Rqt Specification)
  - The key stakeholders, for major requirements, will be specified **explicitly**, along with priority information or reference.
  - Example:
    - **Requirement x:**
      - **Stakeholder X, Y, Z, See X Economics**.

QA 2: All Quality Requirements Quantified

- ! Use quantified multi-dimensional quality requirements
  - to define the *project-relevant* 'Q' in QA.
- !Quality is <u>far more than</u> *bug-freeness!*

You *cannot* ‘Assure’ Qualities that are vaguely defined!

•! **Quantification is an ABSOLUTE PREREQUISITE**

  –! **For Quality Control**

•! **All quality ideas (‘how well’) can always be expressed numerically**

•! **There are many possible quality ‘scales of measure’**

  –! **Some can be quite useful**

  –! **Some are useless**

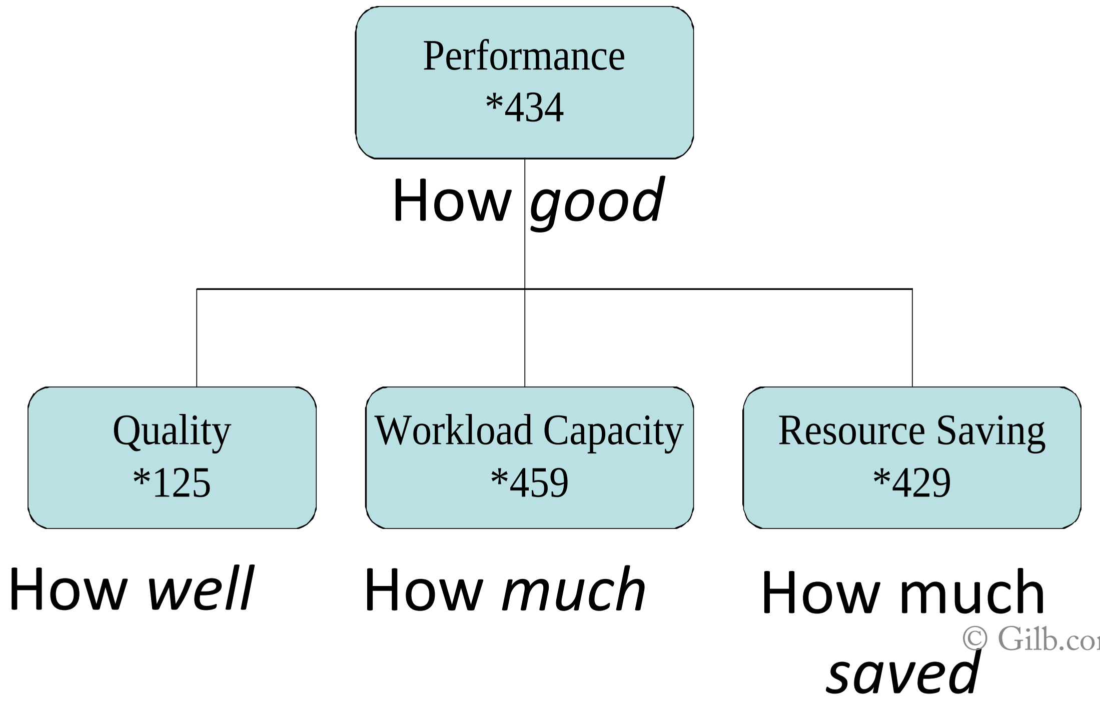
  –! **Some are worse than useless (damaging)**

Quality: the concept, the *noun*

Planguage Concept *125, Version: March 20, 2003

A 'quality' is

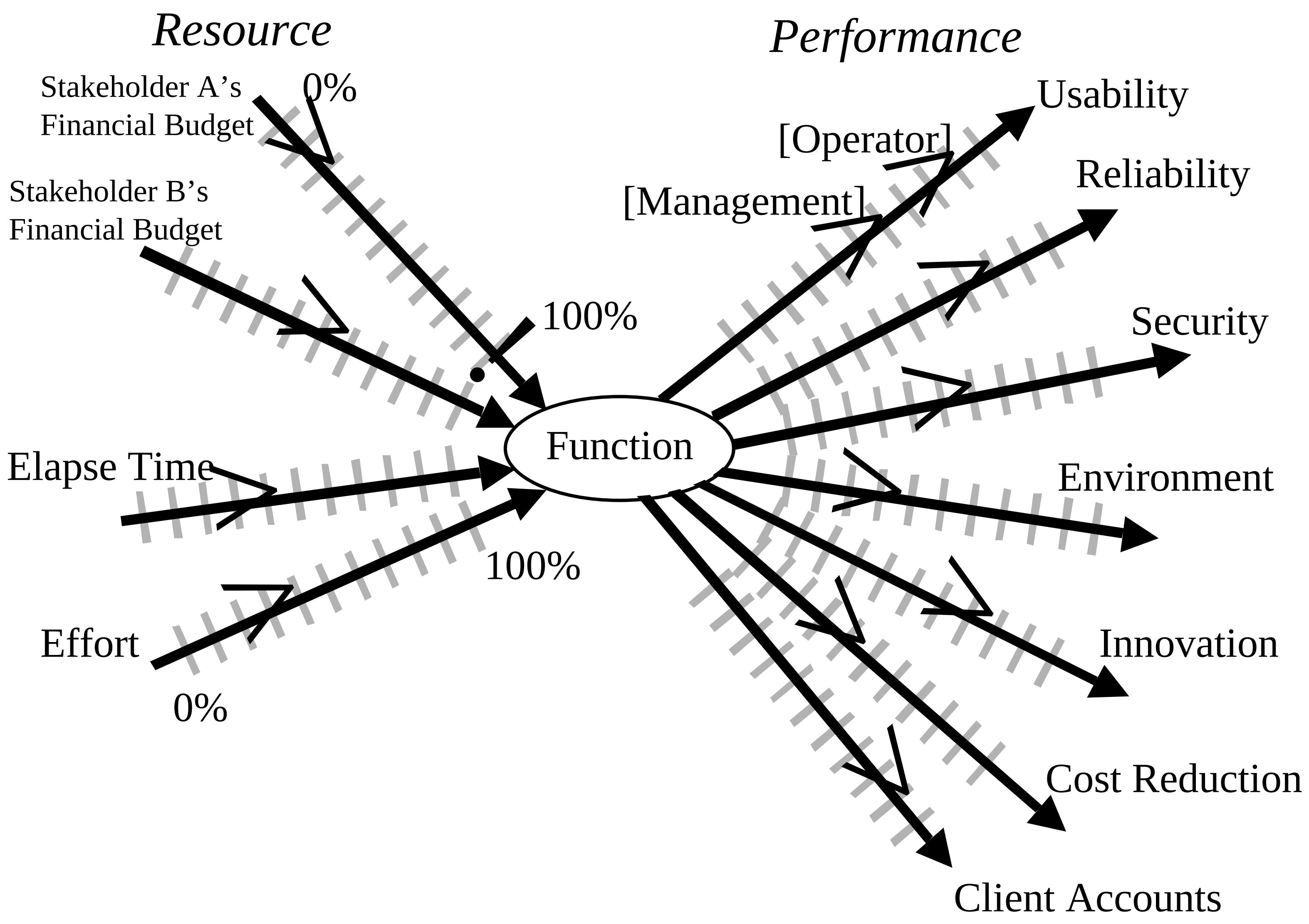
- a scalar attribute -|-|-|-|- (Scale symbol)
- reflecting 'how *well*' ------Past Level<------------>
- a system functions. (Fn)------Past Level<--------->



Some quantification advice

- **Quantify** the *primary* and *critical* qualities
- Quantify ***results***, not means
- Quantify the ***critical*** dimensions, not the 'easy to measure' dimensions
- Quantification is NOT = '***measurement***'
- There are *several quality* ***'levels'***,
  - in space and time,
  - that you will want to put numbers on –
  - not just one number

Multiple Required Performance and Cost Attributes are the basis for architecture selection and evaluation



Quantification Policy

- Quality Requirements and Impacts will ALWAYS be expressed ***numerically***.

- The defined quantification scales will be about the **critical values** and results, not about more easily measurable indirect indicators.

- **Lack** of quantification is regarded as:
  - Incompetent
  - Unprofessional
  - High Risk
  - Not ‘quality’ work
  - Unacceptable, not worth paying for

THE PRINCIPLE OF 'QUALITY QUANTIFICATION'

All qualities can be expressed quantitatively,
'qualitative' does *not* mean unmeasurable.

"In physical science the first essential step in the direction of learning any subject is to find principles of numerical reckoning and practicable methods for measuring some quality connected with it.

I often say that when you can measure what you are speaking about, and express it in numbers, you know *something* about it;

but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind;

it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of Science, whatever the matter may be."

Lord Kelvin, 1893

From http://zapatopi.net/kelvin/quotes.html

© Gilb.com 'Lean QA'

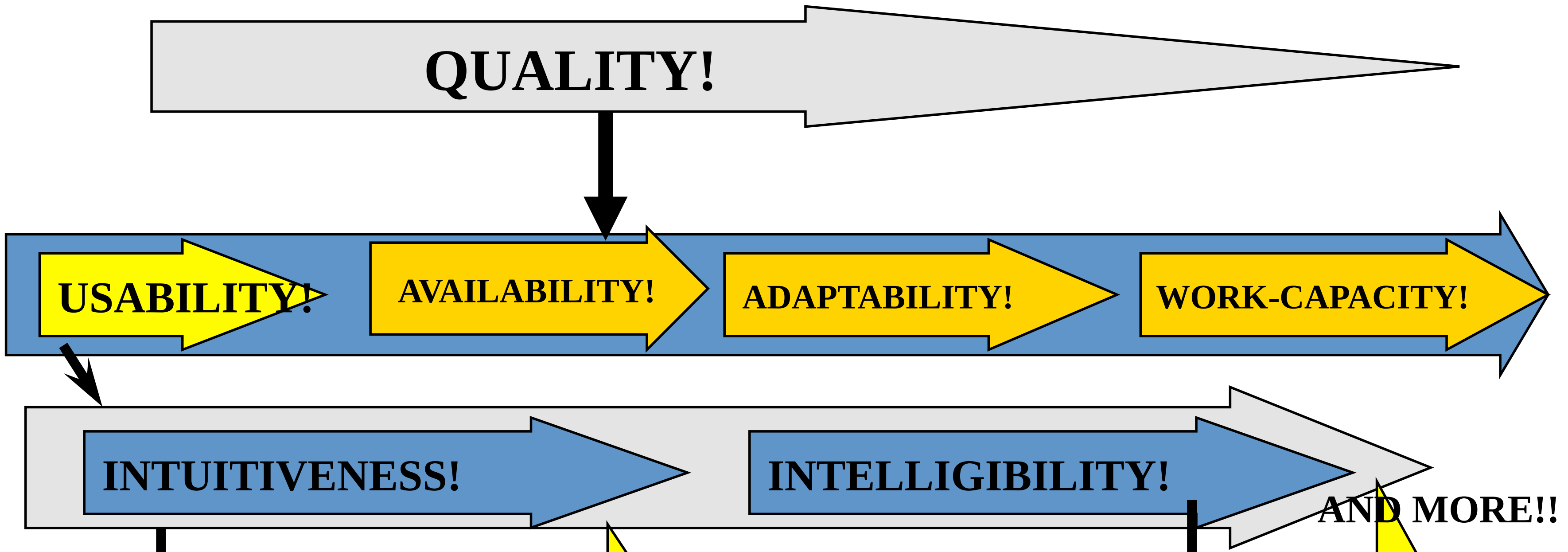
Quantification ‘Rules’

- ***Rules for Quantified specification of requirements and designs***
  - *Examples in ‘Planguage’*
- **All Qualities will be defined by one, or a set of, ‘Scales of measure’**
  - **Scale: Average Correctly Completed Tasks per Hour**
- **Requirements will be expressed as numbers on the scale, and as either Constraints (Fail, Survival levels) or Targets (Wish, Goal, Stretch, Ideal)**
  - **Goal [1st Release] > 20**
  - **Fail <5**
- **Requirement levels will clarify the ‘qualifiers’ [when, where, if].**
  - **Wish [Release 6.0, Asia, Teenagers, If we are in Market] 30**
- **Numeric requirements will be accompanied by related critical information including: source, justification, acceptable ranges.**
  - **Source: Marketing Plan Version 6.5, Page 23-25**
  - **Justification: over 30% additional revenue expected**
- **Quality Estimates (of a design) will be made with information of source, uncertainty, evidence, credibility, risks, issues.**
  - **Strategy XXX is expected to give us 100% of the 1st Release Goal ±40%. Risk: Economic Downturn.**

Quantification Helps…

- Comparative Evaluation
- Quality Requirement Testing
- Project Management
- Deadline Completion Estimation
- Communication of Primary Requirements
- Contracting for results
- Paying Contractors for results
- Reward teams for results achieved
- Motivate Nerds towards Business
- Simplify requirements to Top Ten Critical Ones
- Data Collection & learning
- Research

Quantifying Usability (Erieye C&C System)



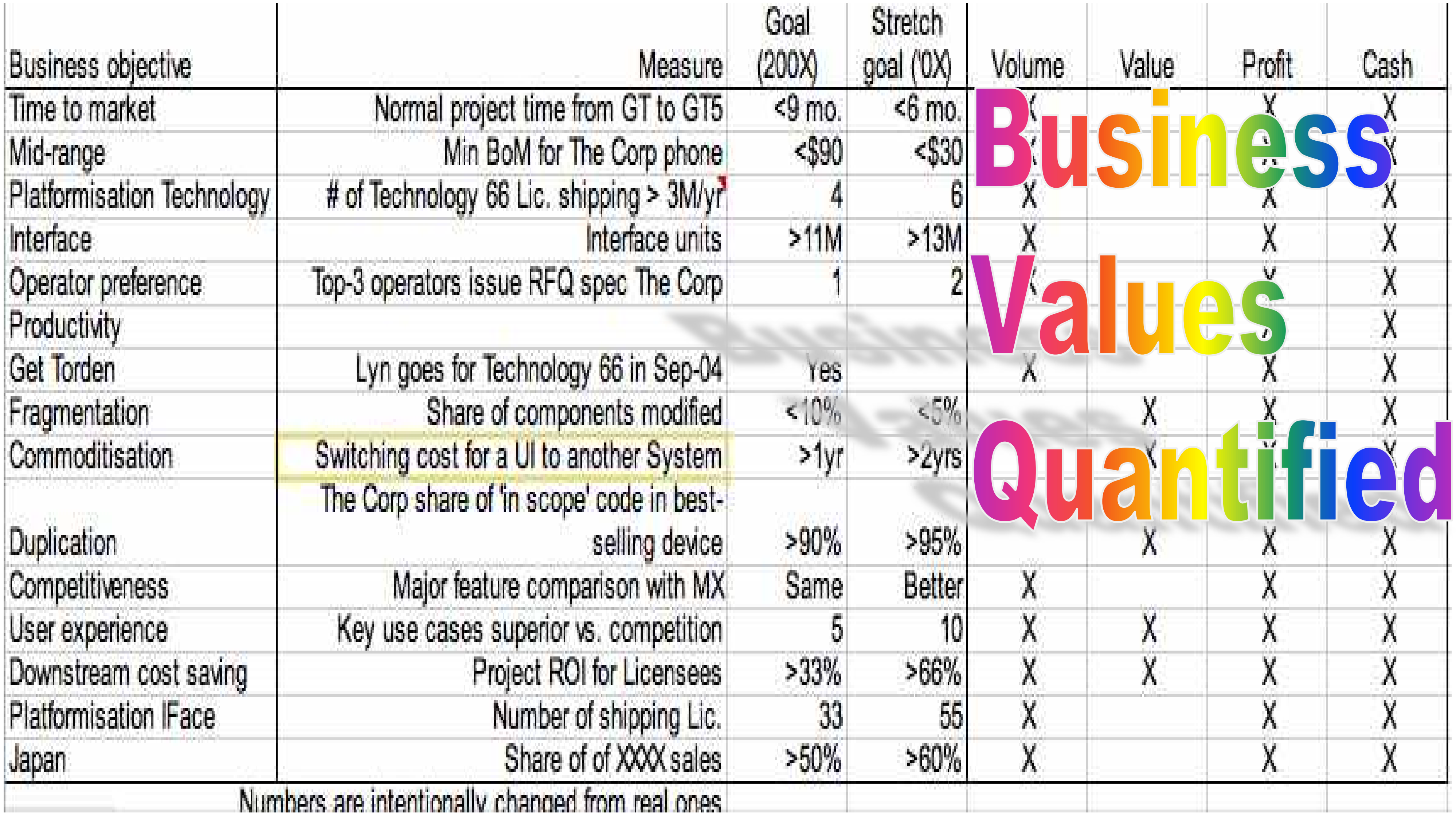
Highly simplified. For detail see CE book, page 162-3

Intuitiveness!
Ambition: Great intuitive capability!
SCALE: Probability that intuitive guess right.!
METER: <100 observations.>!
PAST **[GRAPES]** 80% <-LN!
RECORD **[MAC]** 9%?<-TG!
Fail **[TRAINED**, **RARE]** 50-90%!
Goal **[TASKS]** 99% <-LN!

Intelligibility!
Ambition: Super ease of understanding!
SCALE:% OK interpretations.!
METER: 10 ops., 100 infos, 15 mins.!
P: PAST[20 ops., 300 info, 30 min.]99%!
RECORD **[P]** 99.0%!
Fail **[DELIVERY[1]]**99.0%<-MAB!
!**[ACCEPTANCE]** 99.5%!
Goal **[M1]** 99.9% <-LN!

TRAINED: DEFINED:C&Ctl. operator, approved course, 200 hours duration.!
RARE: DEFINED: types of tasks performed less than once a week per op.!
TASKS: DEFINED: onboard operator distinct tasks carried out.!
ACCEPTANCE: DEFINED: formal acceptance testing via customer contract.!
DELIVERY: DEFINED: Evolutionary delivery cycle, integrated and useful.!

Real Case of top quantified requirements


Business
Values
Quantified

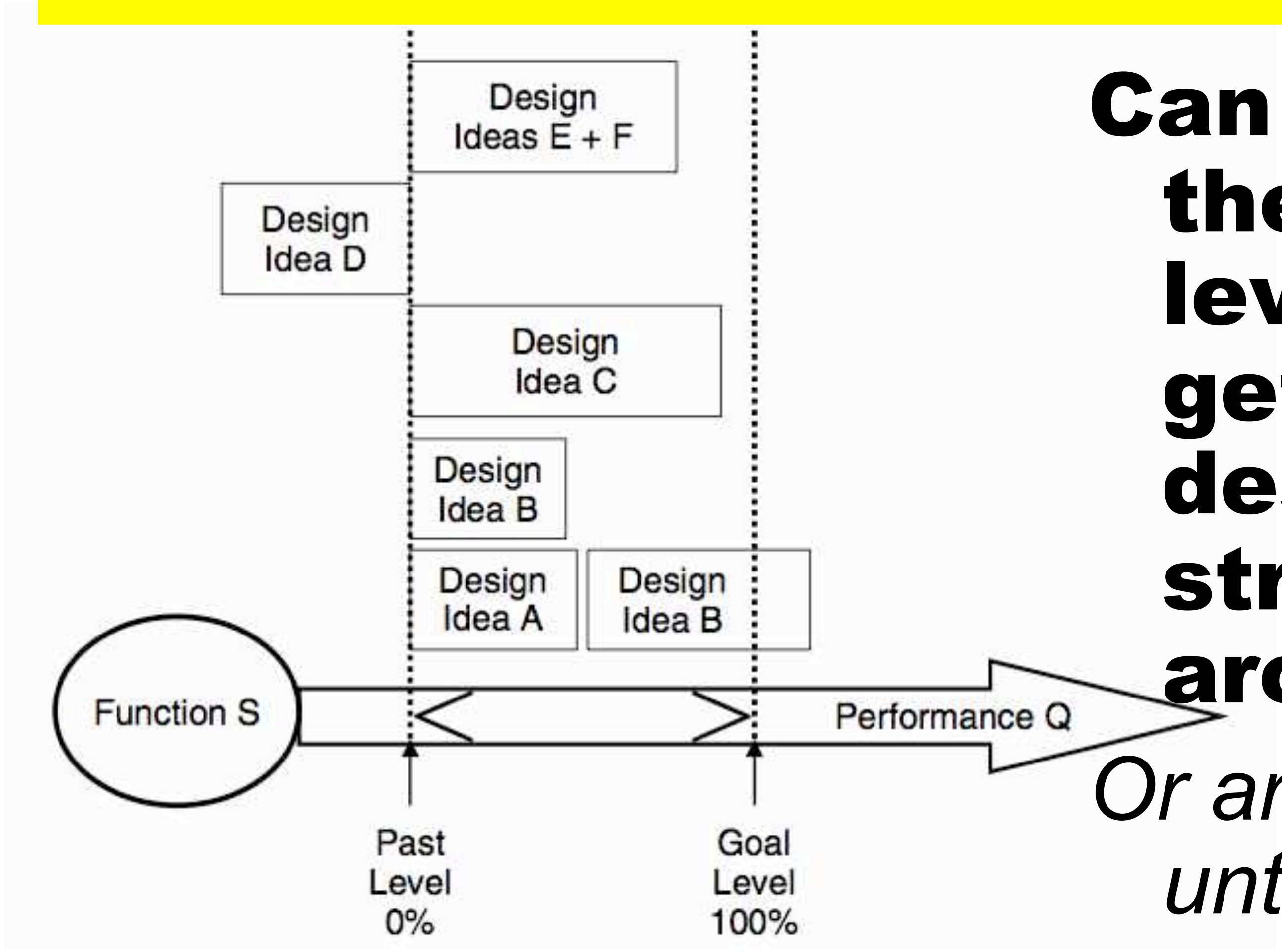
Business objective	Measure	Goal (200X)	Stretch goal ('0X)	Volume	Value	Profit	Cash
Time to market	Normal project time from GT to GT5	<9 mo.	<6 mo.	X		X	X
Mid-range	Min BoM for The Corp phone	<$90	<$30	X		X	X
Platformisation Technology	# of Technology 66 Lic. shipping > 3M/yr	4	6	X		X	X
Interface	Interface units	>11M	>13M	X		X	X
Operator preference	Top-3 operators issue RFQ spec The Corp	1	2	X		X	X
Productivity						X	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

QA 3: Assuring that ***Designs*** *give* Qualities:

- !Estimated **impact of designs** and architectures **on requirements**
  - !*make sure you have reasonable designs before implementation and testing*

Understanding Design Qualities



Can you estimate the quality levels you will get from designs, strategies, and architectures?

Or are you flying blind until landing? ☺

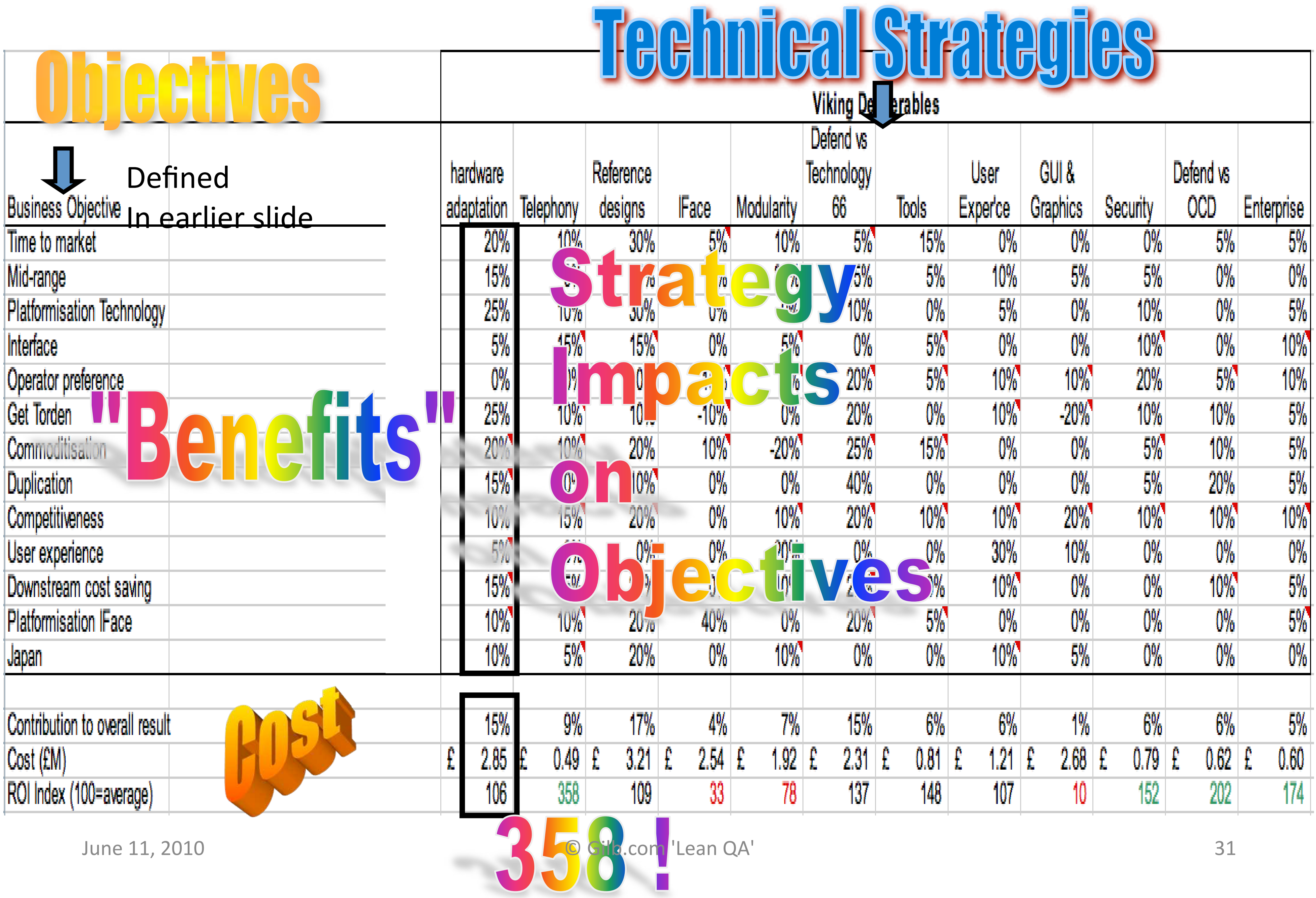
Policy: Design Quality

- ! All professionals, who propose or promote solutions, (Architects, Designers)
  - !Will at least, ***quantify*** **expected impact**, on primary critical quality requirements
  - !and will attempt to **estimate impact** on *other* critical performance, quality, and cost requirements.
  - !and will freely document, and admit**, lack of knowledge about impacts**, on any and all, critical requirements.

Specification Rules: Design Quality

- Use an **Impact Estimation table**,
  - so that we are able to see all relationships,
  - and especially those we have not yet estimated.
- Estimate and specify the designs' impact on the top ten critical objectives:
  - Estimate the incremental contribution to the entire design
  - Estimate ± uncertainty or range
  - Give basis for your estimate (source, evidence)
  - Specify credibility level (0.0 nix -> 1.0 for sure)

Estimating Strategy impact on requirements


Technical Strategies
Objectives
Defined
In earlier slide
Strategy
Impacts
on
Objectives
"Benefits"
Cost
358 !

	Viking Deliverables											
Business Objective	hardware adaptation	Telephony	Reference designs	IFace	Modularity	Defend vs Technology 66	Tools	User Experce	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]	5%	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%	[illegible]	-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%	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

QA 4: AGILE QC:
'Lite' Measurement of Requirements and Specs:

- ! Agile Quality Control (QC) of Specifications (Spec QC).
- !Applies to Requirements, Designs, Codes, Tests
  - to give strong motivation
  - to follow best standards practices
  - better quality by factor of 100.

Assertions:

QC of Specs

- Quality Control needs to **start *early*** (requirements, design)
- Half of your bugs are created **before** coding (GIGO)
- Inspection and Test only find **half** of the problems
- So, 'quality' by 'defect removal' is neither effective, nor **cost-effective**.
- The only cost-effective way to 'get quality' is to
  - **Design** it in
  - **Prevent** defects occurring – by motivation to practice quality development work
- Specification QC (aka 'static testing') can help you 'get quality' in this way
  - Design
  - Prevention

Real EXAMPLE: SAMPLE QC
4.2.2 MEDIA INPUT (127 words). 'C' & 'V' check it

Media Input.

All media rolls, sizes and formats need to be loaded from the front, including heavy weight rolls up to 10 kg. In those "close to the limit cases" (weight or size), an intermediate support may be used to facilitate the roll loading process. Media load experience will be similar for both rolls.

Media information tracking using printed barcodes needs to be supported by the MACHINE*.

Reliability goals for roll switching will be defined in chapter 4.7.

Media cannot be loaded while MACHINE* is not in idle state (error, printing,...)

Roll switching time will be less than 20 sec.

Roll switching time will be as in GW ++. No d-skew algorithms should be used between roll switching except for the first load.

* MACHINEis not real name.

Defect Analysis

(Unclear, ambiguous, design)

4.2.2 MEDIA INPUT (127 words)

- 24 Major Defects by V
- 32 Major Defects by C
- How many defects did the **Team (of 2)** find
  - (or would a team find of 4 people)
  - Guess 32+12 = **<u>44</u>** ±15
- Assuming we found 1/3 (as a team)
  - Total defects existing NOW = 44 (found) + 88 (not found) = **132 ±30?** **(in the 127 words)**
  - **~ 300 Majors/page**
- **Conclusion: we cannot release at Exit level (<1)**

REWRITE EXAMPLE: Media Load Experience (Part of the sample page): (a better design emerged)

Roll Loading Experience

Type: Complex Quality Requirement

Possible Designs: Media Input

Ambition: <Comfort, avoid repetition pain or injury, maximize visibility, flexibility of control, automation choice, info available, similar interface old products, longer time – 2x - between roll loads..> Source: 4.4.2.1 & 4.4.2.2

Roll Loading Experience .Comfort

Type: User Quality Requirement.

Ambition: significantly better than the previous product, and comparable with competitive products.

Scale: minimum distance from wall which permits roll loading.

Past [Old Product, 2009] 60 cm. ?

Goal [New Product. 2010] 10 cm.

Stretch: 0.5 cm.?

Observations and Conclusions/Suggestions from 3 parallel Spec QC exercises

- **Eddy*** (found 24 Major defects)
  - **Primary objectives need to be clear**
  - **This idea is not easily bought into**
- **Vince** (found 32 Defects)
  - **Unbelievable level: I would have expected much lower**
  - **Much room for improvement**
- **Louis 28 Majors in 307 words**
  - **These 'guidelines' are not 'requirements'**
  - **We need to analyze our truth of damage downstream, from major defects, in different spec types.**
- **Chris 59 Majors in 307 words**
  - **There are too many redundant words, 'waste'**
  - **Exit level needs to be tuned to spec type**
- **Jack: 66 Majors in 464 words**
  - **"We face 2,000 bugs at cost of 20x more effort to fix downstream**
  - **In this project, we went through 4,000 to 8,000 bugs**
  - **Conclusion: the experiment is close to reality"**
- **Charley: 90 Majors found in 464 words**
  - **We have learned the impact of a document that initially appeared to be OK**
  - **We need to address this situation NOW,**
    - **Not optional or nice to do!**
- * not real names

Agile Spec QC:
What is it?

- **Critical** documents are checked for *conformance to 'rules'*
- **Rule violation** = '<u>defects</u>' (law violation = illegal acts)
- We ***<u>sample</u>*** large documents, to avoid high cost of measurement
- We try to **measure** 'defect density'
  - Major defects per page
- We accept documents that meet our **'exit conditions'**
  - Like: "Maximum 1.0 Majors/ page remaining"
- The result is 'no garbage out', '**no garbage in**'
- **Not** because we remove 'garbage'!
- But because people are really strongly **motivated** to follow the best practices, in the rules.
- Otherwise, they are clearly Not doing acceptable **professional** work
  - and they want to feed their kids ☺

Expectations: SQC

- **<u>Defects reduced </u>**by about 10 x in 6 months
  - By another 10 x by concerted long term effort in 2-3 years
- Individuals will go through a **<u>steep learning</u>** curve
  - 50% reduction in their injected defects per cycle of learning
- The organization will begin to take their **standards** (rules) **seriously**

Policy: SQC

This is the **most important thing;** I wish **managers** would implement.

- !All **critical specifications** will be ***measured*** for defects
- !**Defective** work, *over the exit level*, will ***not*** be **released** for others to use

Summary: SQC

- Spec QC – 'static' test is more far **cost effective** than testing
- With testing, when the stream of injected defects is constant, test finds a **small %** of them, example 30%±
- With Spec QC – we drive the defect **injection down** towards zero

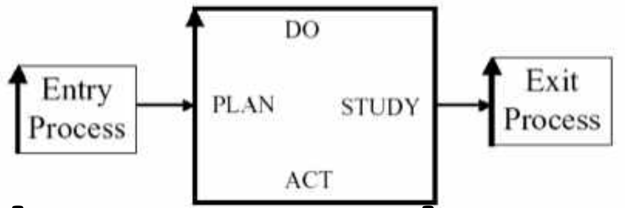
The formal Agile SQC Process Sources

- Cutter 5 pg Paper
- http://www.gilb.com/tiki-download_file.php?fileId=64

- INCOSE SQC Paper
  http://www.gilb.com/tiki-download_file.php?fileId=57

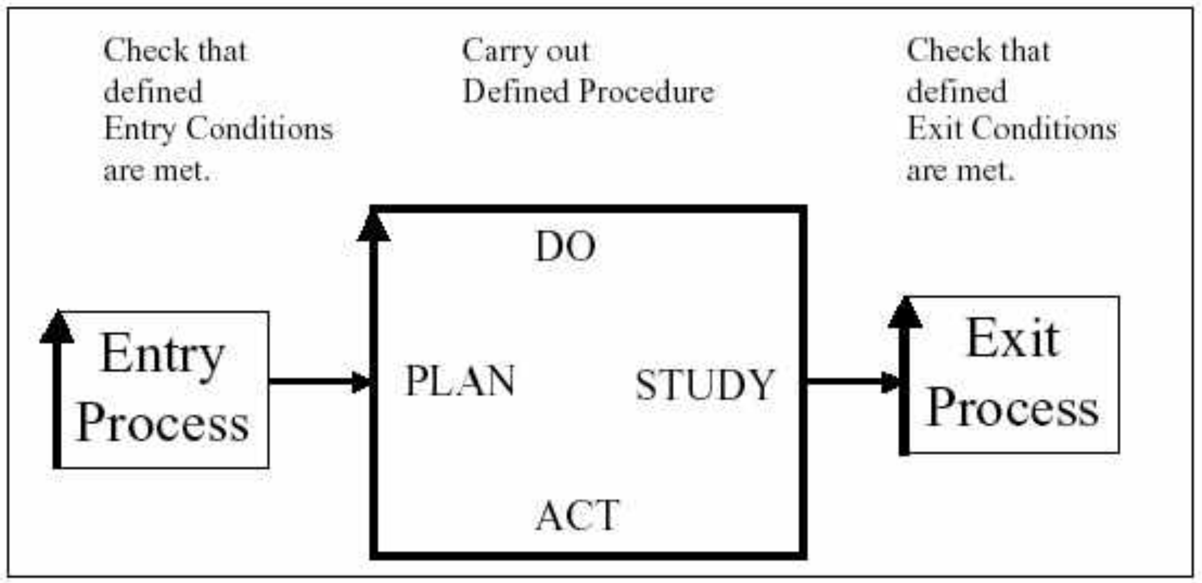
- Agile SQC Slides with Standard for Process
- http://www.gilb.com/tiki-download_file.php?fileId=239

QA 5: Quality *Gateways* to the Work



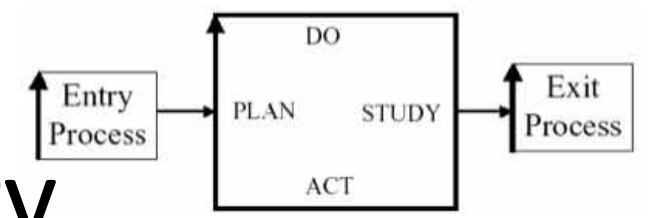
- ! Process Entry and Exit numeric standards
- ! 100x improvement
- !for compliance to specification standards:
- !no Garbage In Please.

You should have NUMERIC exit and entry quality levels from both test processes and related development processes

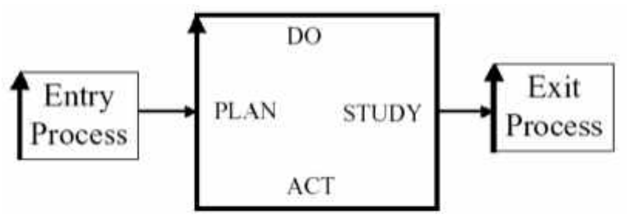

Check that defined Entry Conditions are met.
Carry out Defined Procedure
Check that defined Exit Conditions are met.
Entry Process
DO
PLAN
STUDY
ACT
Exit Process

- Entry and Exit Condition example:
- Maximum estimated 1.0 Major defects per logical page remaining.
- This was the MOST important lesson IBM learned about software processes (source Ron Radice, co-inventor Inspections, Inventor of CMM)
- No ‘Garbage In’ to Test Planning!

Assertions: Exit/Entry



- ! Numeric Process Exit (and Entry) control is one of the simplest, yet most powerful, quality control ideas, in practice
  - !IBM's experience (Ron Radice – CMM/Inspection)
- ! Almost nobody has managed to learn and apply this simple universal idea
  - !Most all organizations have terrible Garbage Out problems, like 100 majors/page out
  - !But they don't know that!
  - !No measurement. No control.

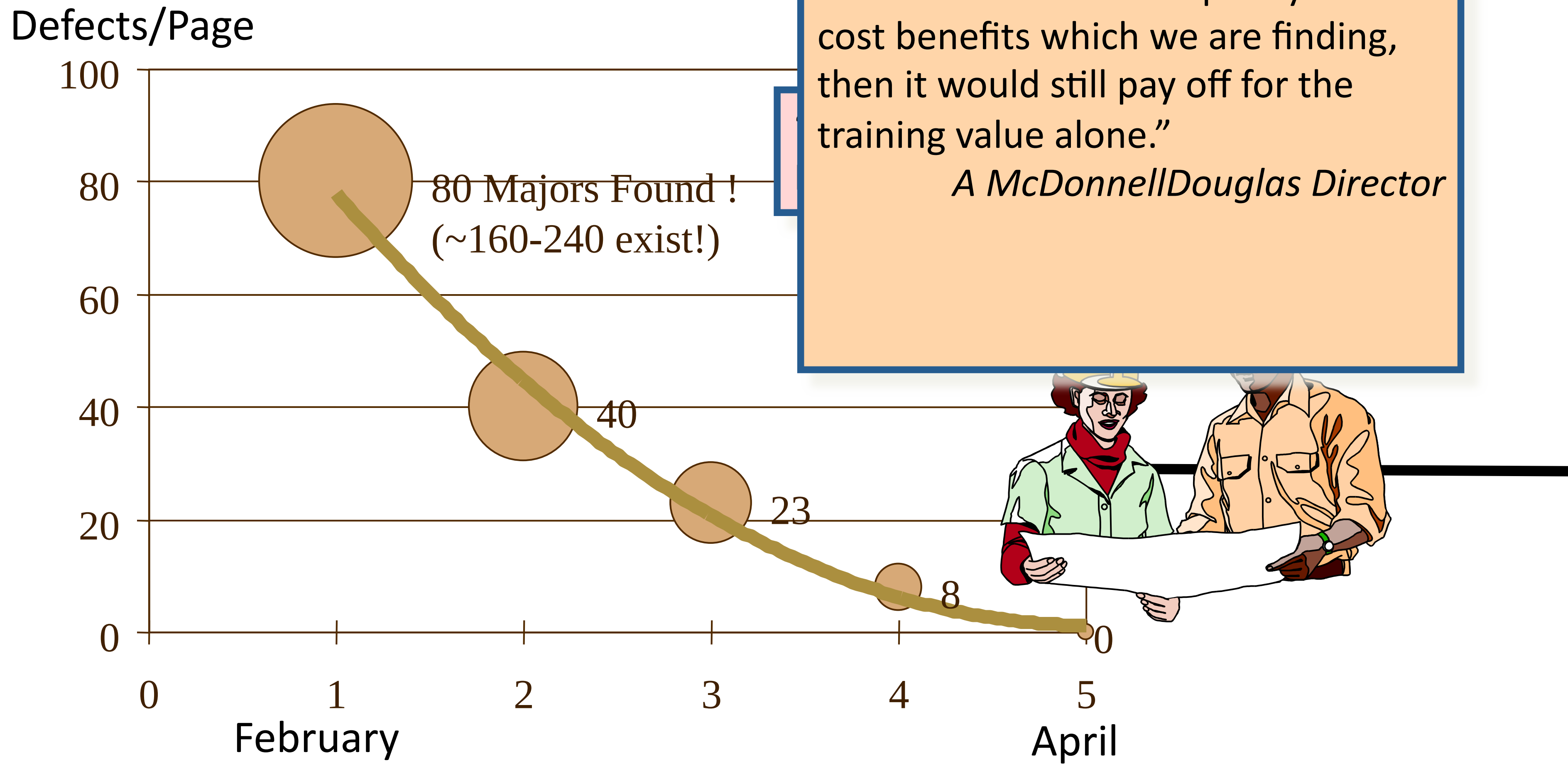


Basics: Exit Entry

- ! Process Exit (example from a test planning process)
  - ! Depends on **a set of Exit Conditions**
  - ! The key condition is **'remaining defect density**' (<1/p ?)
  - ! The exit level is ultimately set '**economically**'
    - ! It pays off to exit now
  - ! The short term effect (today) is to **prevent bad stuff** happening
  - ! The long term (next week) is to **motivate people** to do their job professionally
    - ! And radically (100x !) reduce their defects injected

Positive Motivation

Personal Improvement



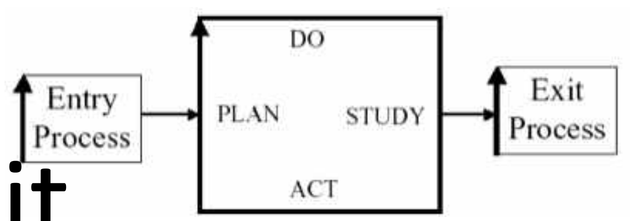
Inspections of Gary's Designs

"We find an hour of doing Inspection is worth ten hours of company classroom training."

A McDonnell-Douglas line manager

"Even if Inspection did not have all the other measurable quality and cost benefits which we are finding, then it would still pay off for the training value alone."

A McDonnellDouglas Director



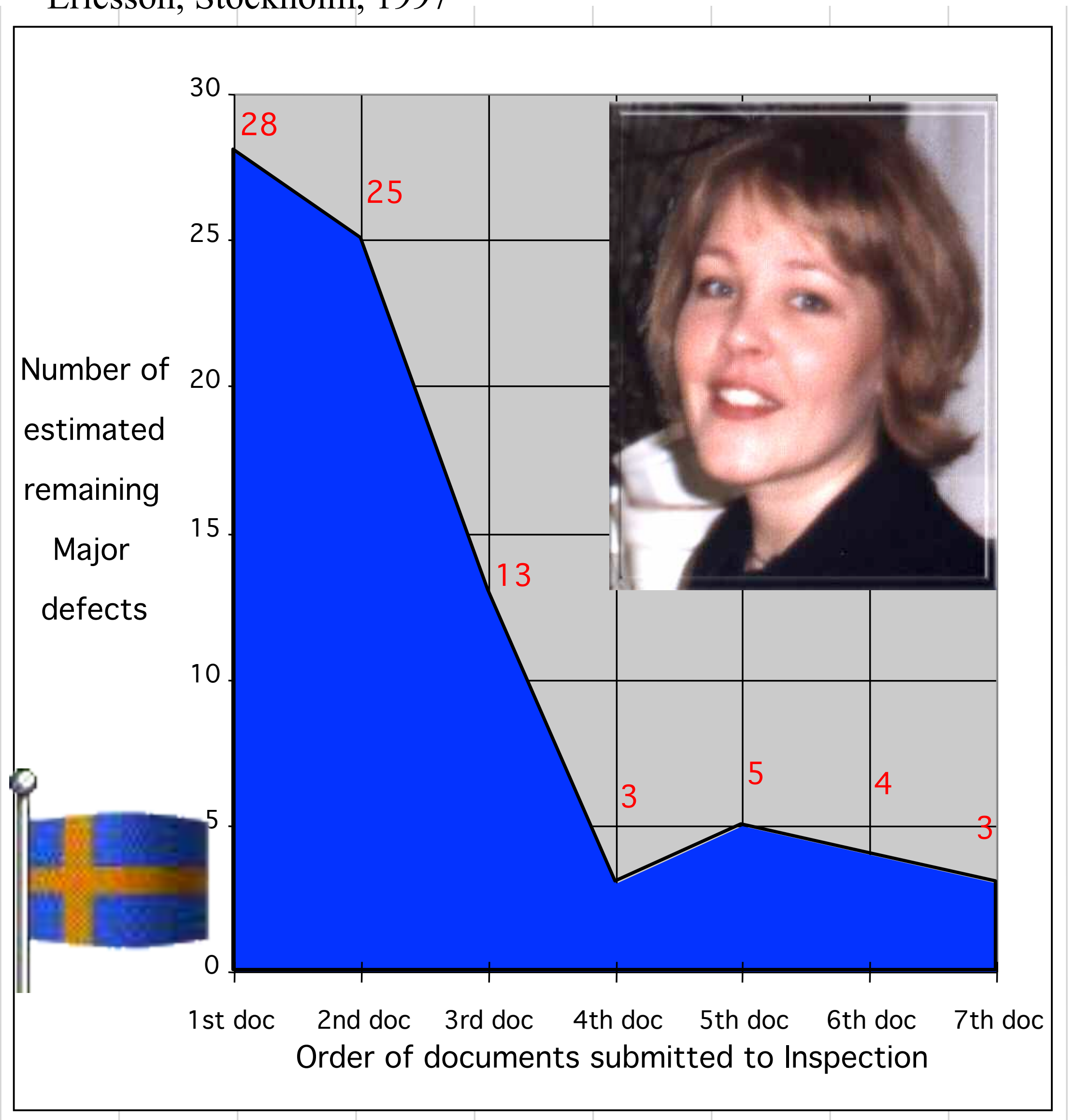
Expectations: Exit

- ! **If** you apply numeric, quality level, exit conditions to a process – you can expect:
  - !Drastic **reduction** in (defective work) Major **Defects** (10x, then 100x)
  - !Developers will bother to **learn**, and to **apply** correctly – basic 'rules' (like 'clear')
  - !Each individual will have to go through a gradual **process of learning** the 'rules'
    - ! Getting 50% better each time they try to met the exit level

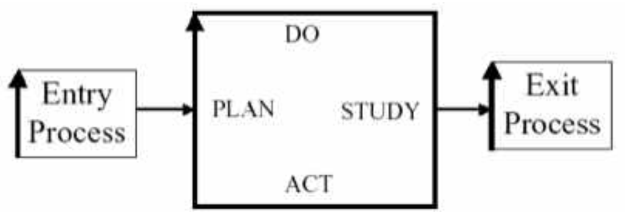
Individual learning Curve

- •! Individual Learning Curve
  - –! The speed which the individual learns to follow the Rules,
  - –! As measured by reduced Major Defects found in Inspections
    - •! Notes:
      - –! Faster, earlier and more dramatic than "process improvement"
      - –! Never mentioned in literature as a measurable

Marie Lambertsson's Learnability Curve, Ericsson, Stockholm, 1997



See also the Raytheon Learning Curve



Policy: Exit

.

- All important work processes will be **managed** by formal written **exit conditions**
  - The most powerful and objective condition will be based on the level of major defect density
  - The minimal standard will be ‘less than 10 majors per 300 words (a virtual page)’, but ultimately < 1 major/page
  - A defect is violation of your written ‘rules’
  - A ‘Major’ defect, is a rule violation potentially impacting quality of the final product.
- **Managers** will be held **personally responsible** for all **bad quality**, and for and **schedule slippage**;
  - due to work they have accepted, which is *worse* than this standard.

Defect Rates
in 2003 Pilot Financial Shop, London, Gilb Client
Spec QC/Extreme Inspection + Planguage Requirements

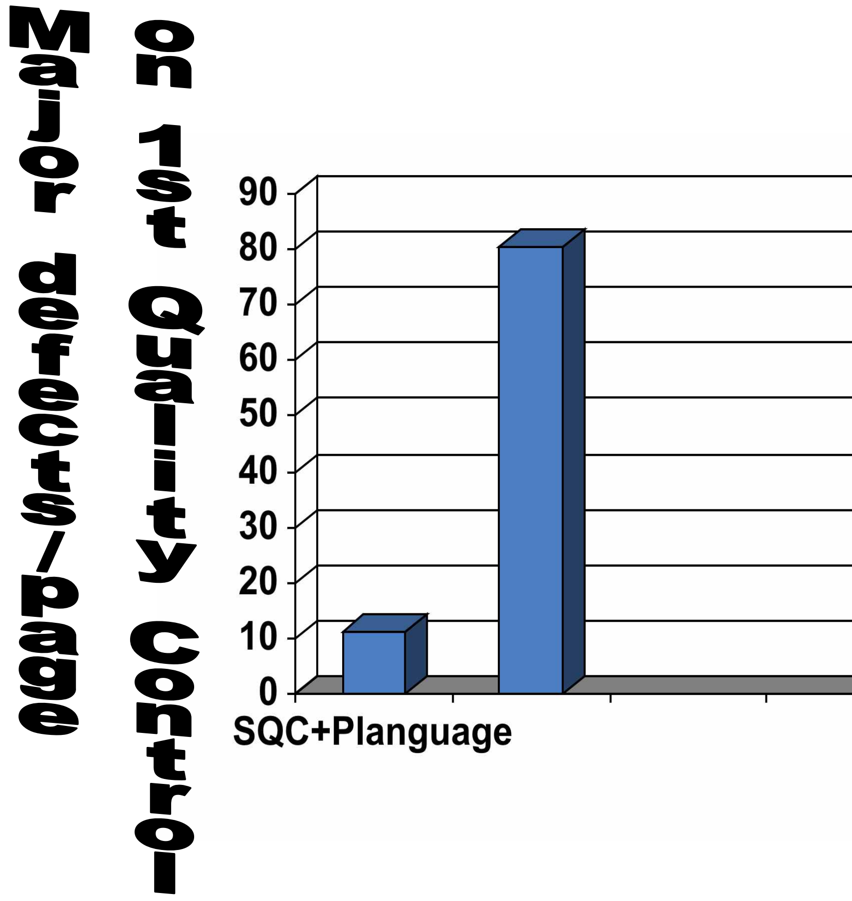
Across 18 DV (DeVelopment) Projects using the new requirements method, the average major defect rate on first inspection is 11.2.

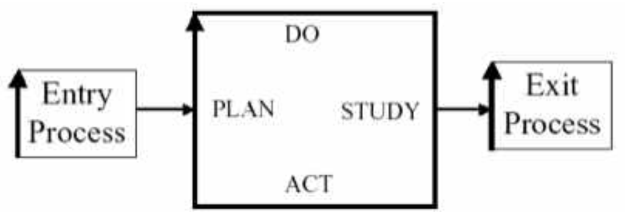
4 of the 18 DV projects were re-inspected after failing to meet the Exit Criteria of 10 major defects per page.

A sample of 6 DV projects with requirements in the 'old' format were tested against the rules set of:

> **The requirement is uniquely identifiable**
> **All stakeholders are identified.**
> **The content of the requirement is 'clear and unambiguous'**
> **A practical test can be applied to validate it's delivery.**

The average major defect rate in this sample was 80.4.





Policy: Entry Conditions

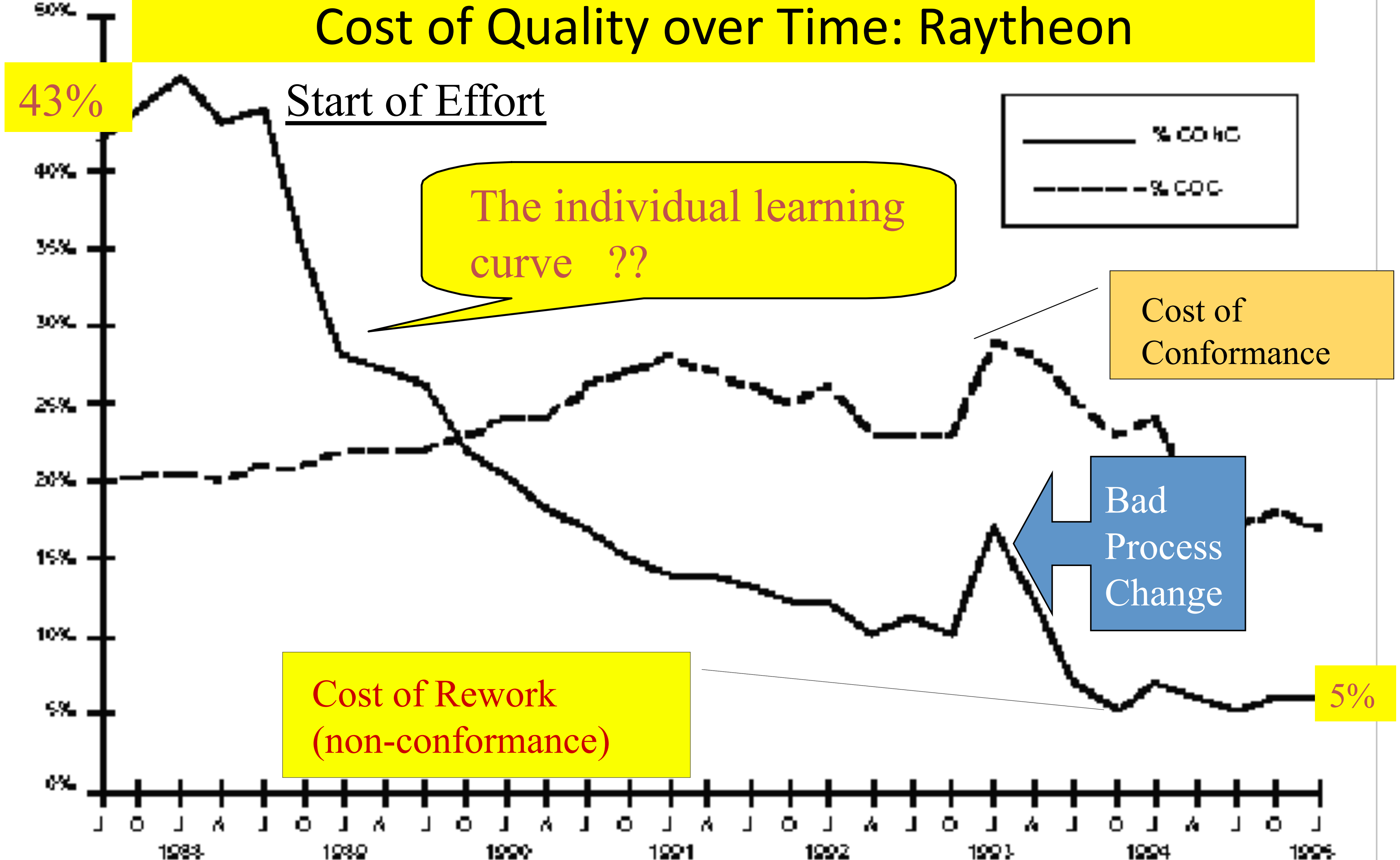
- Each work **process** (example test planning, using requirements as inputs) must '**defend**' themselves against 'garbage' input, **by** a formal set of **entry** conditions.
  - One *entry condition* is the level of *major defect density*.
  - Anything over 10 majors/page is *always* unacceptable
    - But most of you now start at the 80-280 M/pg level!
  - Ultimately, '**high quality**', means *less* than 1 Major/page
  - The Entry Process is not obliged to accept claims from the Exit process proceeding it.
    - If necessary, the entry process should at least do random 1-page sample, Spec QC, to check quality levels.

QA 6:

Defect Prevention Process 'DPP': CMMI 5

- !Measurable Process Improvement
  - !early, frequently, practical (grass roots!),
  - !in all types of projects.

Cost of Quality over Time: Raytheon



www.sei.cmu.edu/publications/%20documents/95.reports/95.tr.017.html

End 1988

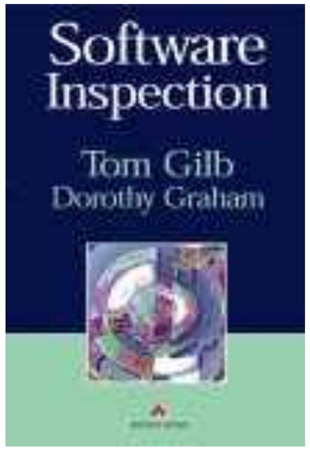
End 1994

Figure 8: Cost of Quality Versus Time

© Gilb.com 'Lean QA'

Assertions about DPP (Invented by Robert Mays, IBM

- DPP is similar in principle to **Deming's Plan Do Study Act** – Statistical Process Control (SPC), and to Six Sigma (GE, Motorola).
- DPP has a return on investment of about **13 to 1**
- DPP will **reduce injected defects** for any work process **by 2/3** in the first year or two, **and far more** in the longer term (approaching zero defects)
- DPP is far **more cost effective** than any known form of Inspection or Testing – in delivering quality



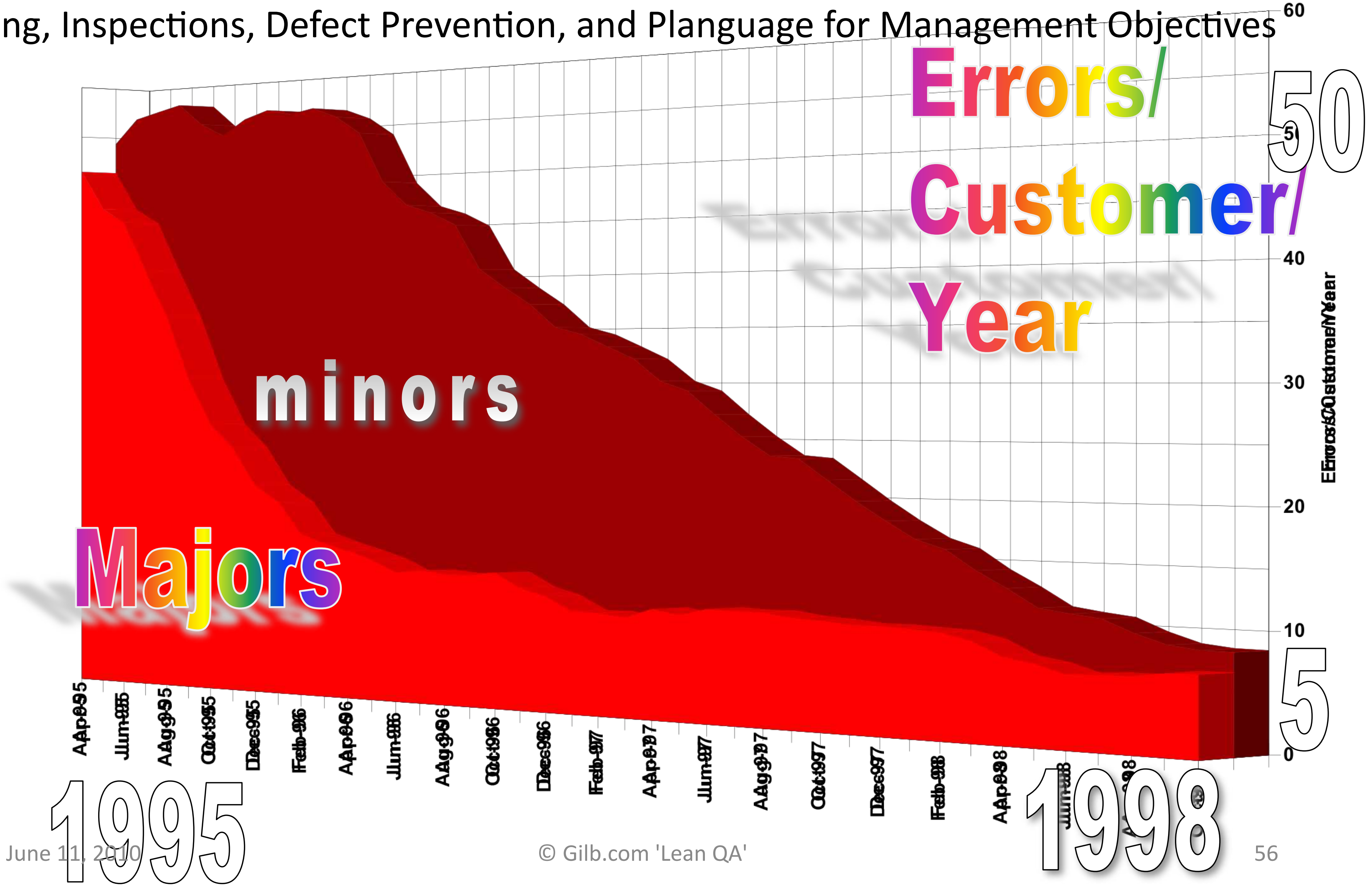
DPP = Chapter 7 & 17

Improving the *Reliability* Attribute

Primark, London (Gilb Client)

see case study Dick Holland, “Agent of Change” from Gilb.com

Using, Inspections, Defect Prevention, and Planguage for Management Objectives

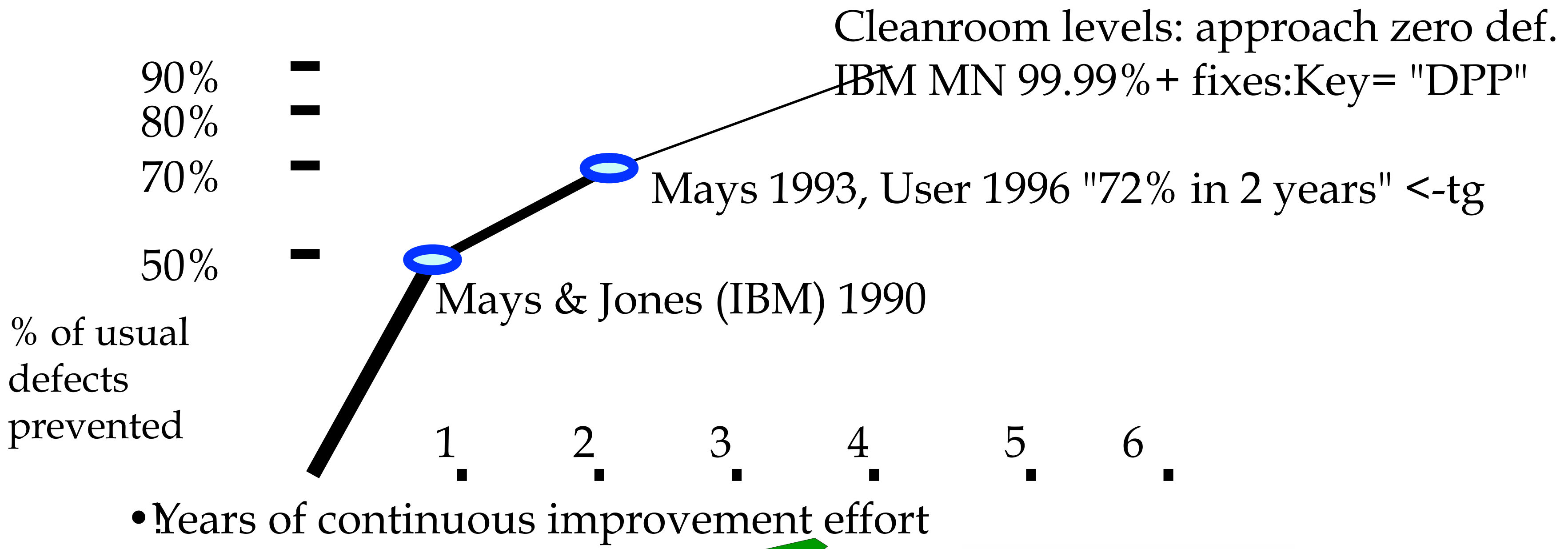


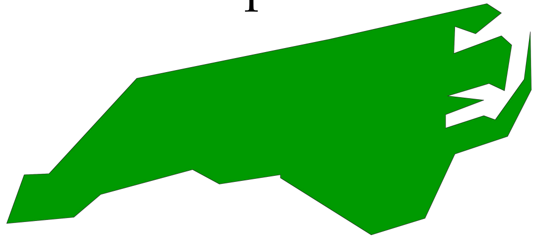
What *is* DPP?

- **Grass roots developers** (checkers, testers) analyze frequent-type real bugs – *concrete detailed instances that they have first hand knowledge about* - or major defects
- **Developers** develop local personal or group **opinions** about *root causes*, and potential '**cures**'
  - changes in process, or anything that might reduce defect occurrence)
- **Developers** try out *their ideas* in **practice**,
  - locally (*their* project or product, and *their* process, as practiced by *them*)
- Ideas that **work**, are picked up at a higher level of corporate process change, and implemented **corporate wide**

Story: Tom and the young Douglas Aircraft Engineers – the cut outs solution. Simple!

Defect Prevention Experiences: Most defects can be prevented from getting in there *at all*



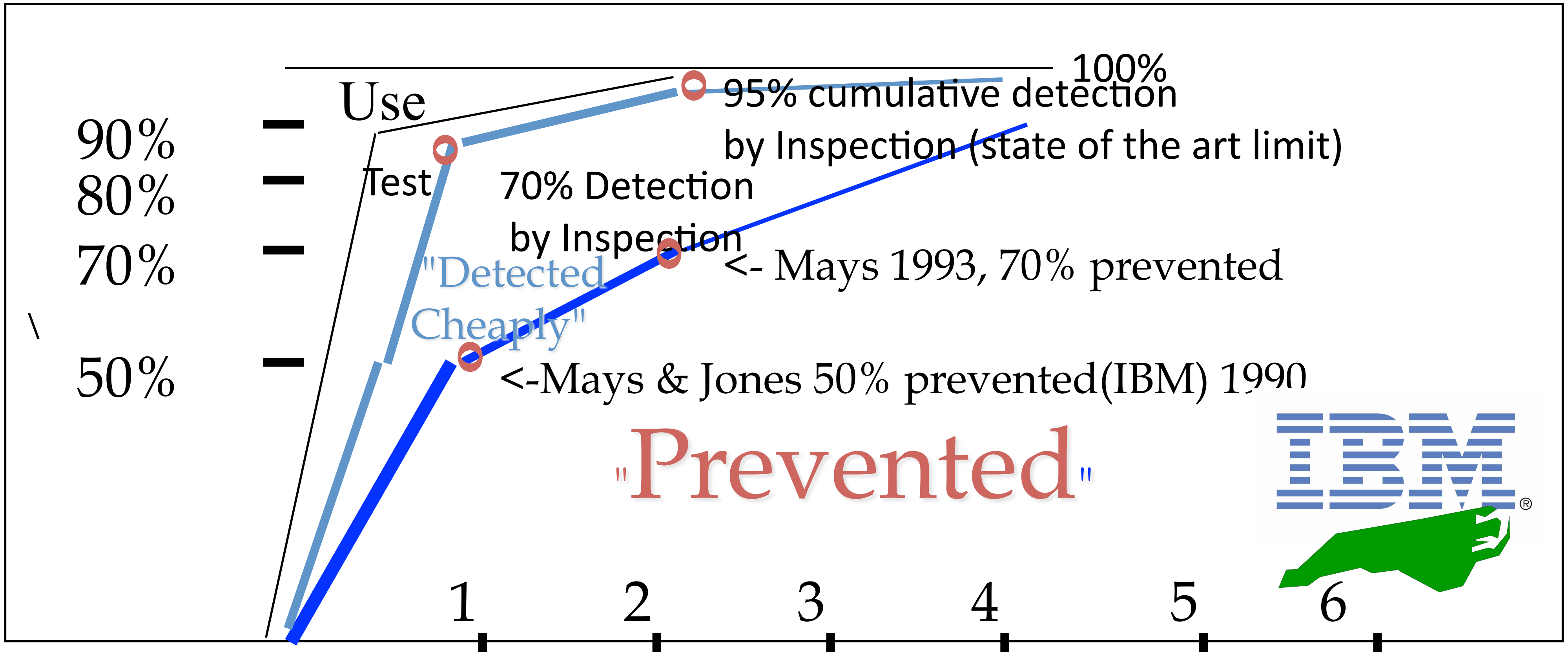

North Carolina

IBM Research Triangle Park Networking Laboratory

How does DPP work?

- After an Inspection/Spec QC Process, **Major defects are examined** by the checking team. Half an hour sessions.
  - They are looking at a colleagues work, colleague is there (the source of defects: knows why)
- They arbitrarily select one, of a small group of **recurrent** types of **defects**, to work on (3 minutes each). 10 in 30 minutes.
- They **brainstorm root causes** (organizational, not personal)
  - Like: *misleading training course information*
- They **brainstorm possible 'cures'**
  - Like: *enhance slides, and tests to make the point clearer.*
- *They may themselves**, carry out the proposed changes** and **try them** to see if they work. Keep it simple – prove concept works.*
- ***Successful** changes are picked up at **corporate** quality level and instituted more **widely** and more **properly**.*

Prevention + Pre-test Detection is the most effective and efficient



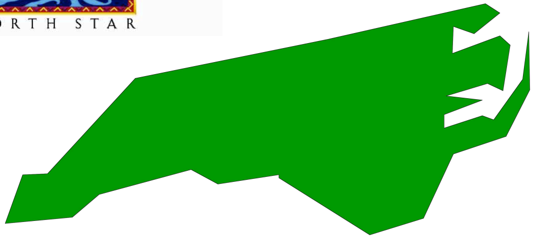
- Prevention data based on state of the art prevention experiences (IBM RTP), Others (Space Shuttle IBM SJ 1-95) 95%+ (99.99% in Fixes)
- Cumulative Inspection detection data based on state of the art Inspection (in an environment where prevention is also being used, IBM MN, Sema UK, IBM UK)

Effects of DPP: Grass roots wisdom

- Systemic (due to **'common cause') defects** are reduced quickly and in volume (2/3 in year)
- **The i*nside knowledge* of local teams** is exploited – how things really work in the real world – why the defects *really* occurred
- The *inside local understanding* **of *socially acceptable changes*** is used: people will not suggest changes they would hate to do themselves
- The feeling of **'empowered creativity'** to find process improvements that *really* work, is very motivating to the grass roots professionals!
  - Big costly ideas that *never wor*k, as often suggested by management, architects, and interested suppliers, are not imposed on the developers.
  - Ideas that don't work are discarded quickly, or re-tuned to work better.
- Many **small but practical improvements**, quickly and cheaply deployed, 200-2,000 annually, add up to major measurable change in quality.
- Any one group (like 'test', or a 4 person development team) can use this method on their *own work*, to prove how well it works – improving quality.

IBM MN & NC DP Experience

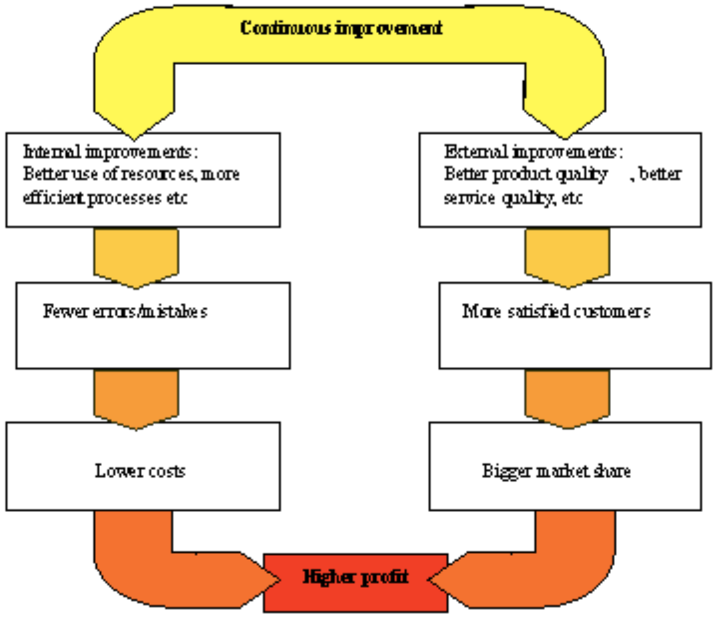
- 2162 DPP Actions implemented
  - between Dec. 91 and May 1993 (30 months)<-Kan
- RTP about 182 per year for 200 people.<-Mays 1995
  - 1822 suggested ten years (85-94)
  - 175 test related
- RTP 227 person org<- Mays slides
  - 130 actions (@ 0.5 work-years
  - 34 causal analysis meetings @ 0.2 work-years
  - 19 action team meetings @ 0.1work-years
  - Kickoff meeting @ 0.1 work-years
  - TOTAL costs 1% of org. resources



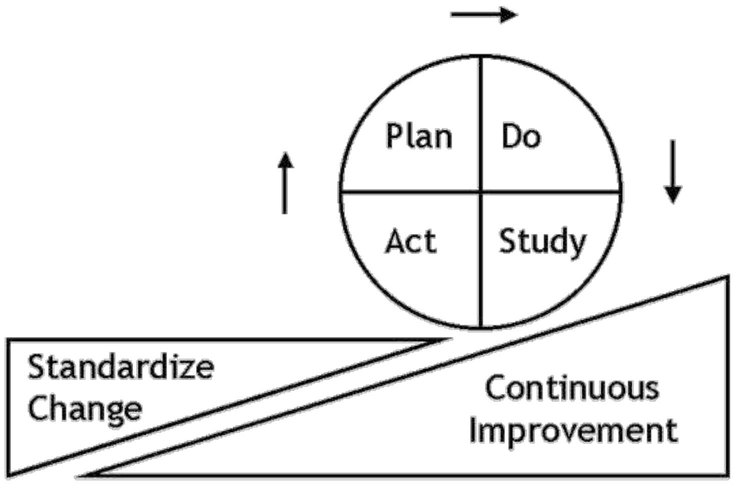
- ROI DPP 10:1 to 13:1, internal 2:1 to 3:1
- Defect Rates at all stages 50% lower with DPP

DPP Policy

- All interested teams will be given **regular opportunities** to analyze their own defects using the DPP process.
- They will be given the **opportunity to try out** their solutions; and *measure* the *effects* of their solutions.
- Local **successful** solutions will be **adopted more widely**, by groups responsible for change and improvement (CTO level)
- Local groups responsible for initially finding, and successfully trying out improvements will be **suitably honoured and rewarded**.
- The **minimum amount of annual investment** in this activity is **5%** of total work hours.
- **Failure to successfully invest** in this each year, irrespective of excuses*, will be considered a serious **management failure**.
  - * 'meeting deadlines' is an invalid excuse. Deadlines are a major reason for doing this properly. Defects destroy deadlines!
  - Investments early in critical projects can easily save those projects.


Continuous improvement
Internal improvements: Better use of resources, more efficient processes etc
External improvements: Better product quality , better service quality, etc
Fewer errors/mistakes
More satisfied customers
Lower costs
Bigger market share
Higher profit

Prevention Costs



Deming Cycle

- •! 5%, stable at 5%
  - –!of development costs
  - –!(Raytheon 1993)
- •! 0.5 % of development costs
  - –!(Mays 1995)

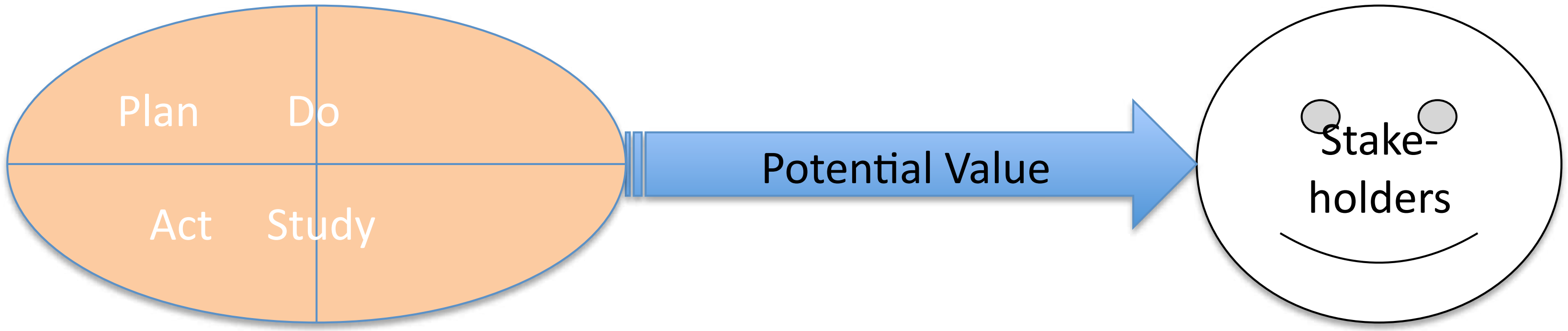
Defect Detection strategies versus Defect Prevention strategies

- Defect **Detection**
  - (inspection, **test**, customer reports)
  - Is *ineffective* for getting high bug-freeness into systems
  - It is better than nothing
  - Inspection is cheaper than test-and-debug
- Defect **Prevention** - is at 2 levels
  - process improvement
    - (CMMI Level 5)
  - individual capability improvement
    - (50% per motivated cycle)
- Defect prevention is **BY FAR the smartest** one

QA 7: Rapid Evolutionary iteration: ‘Evo’

- do real QA weekly,
- and incrementally.
- Incremental value delivery, data collection, feedback, analysis and change:
- for early value delivery,
- or cost control,
- for intelligent prioritization,
- for team and process validation of effectiveness.

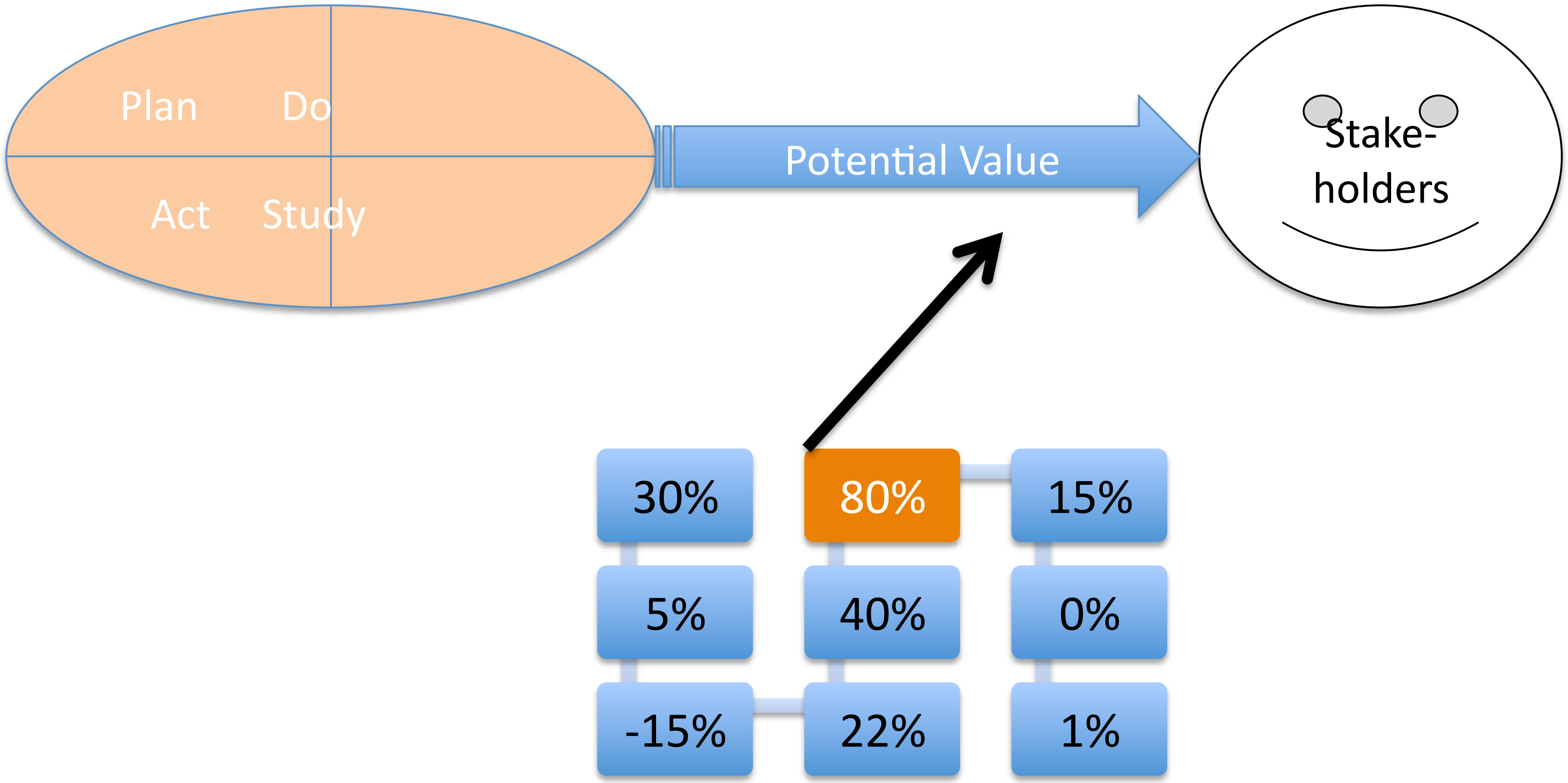
Primary Evo Concept: Deliver *Potential* Value



The Evo Cycle:
Viewed as a Deming PDSA Cycle

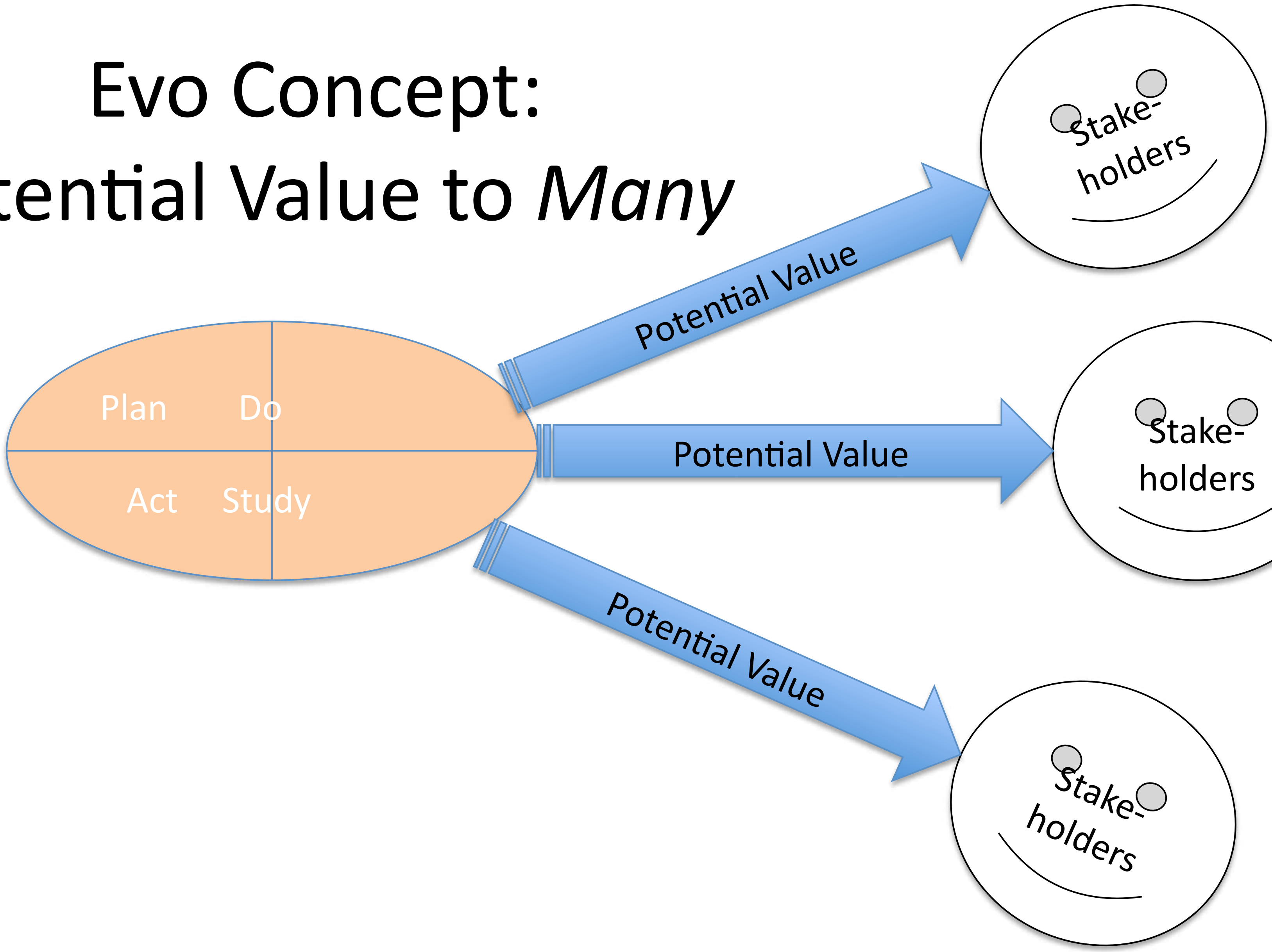
- Incremental Value Delivery to Stakeholders

Deliver the highest value for resources



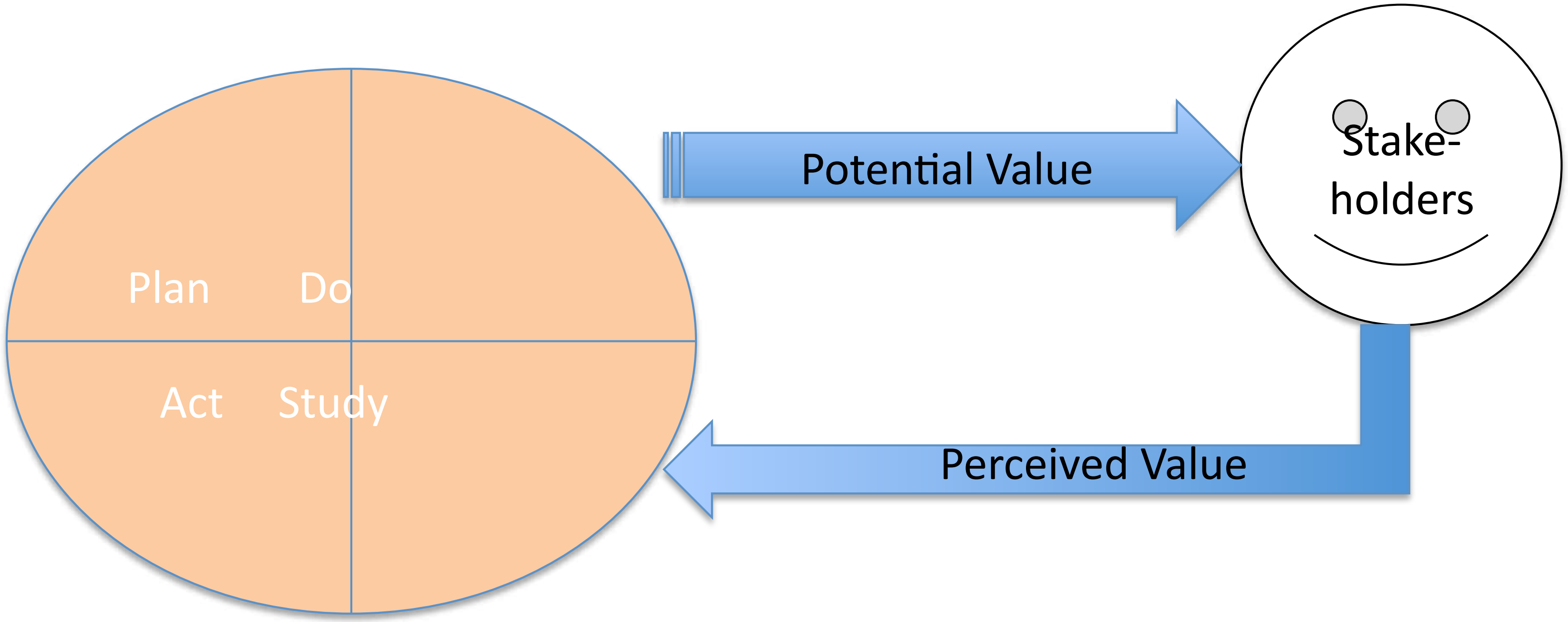
HIGHEST AVAILABLE Incremental Value Delivery to Stakeholders

Evo Concept: Potential Value to *Many*



•! Incremental Value Deliveries to *Many* Stakeholders

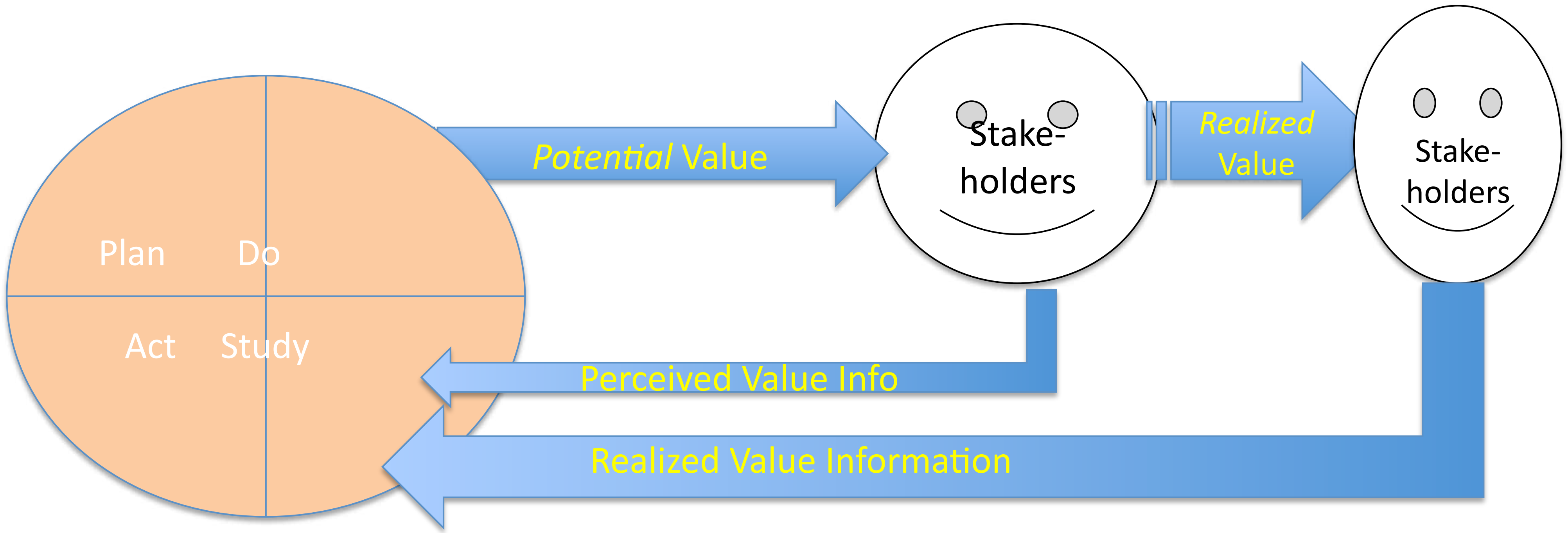
Evo Concept: Short Term Feedback
"This looks like a change I can get value from!"



- ! Initial Feedback from Stakeholders, after Evo Cycle delivery

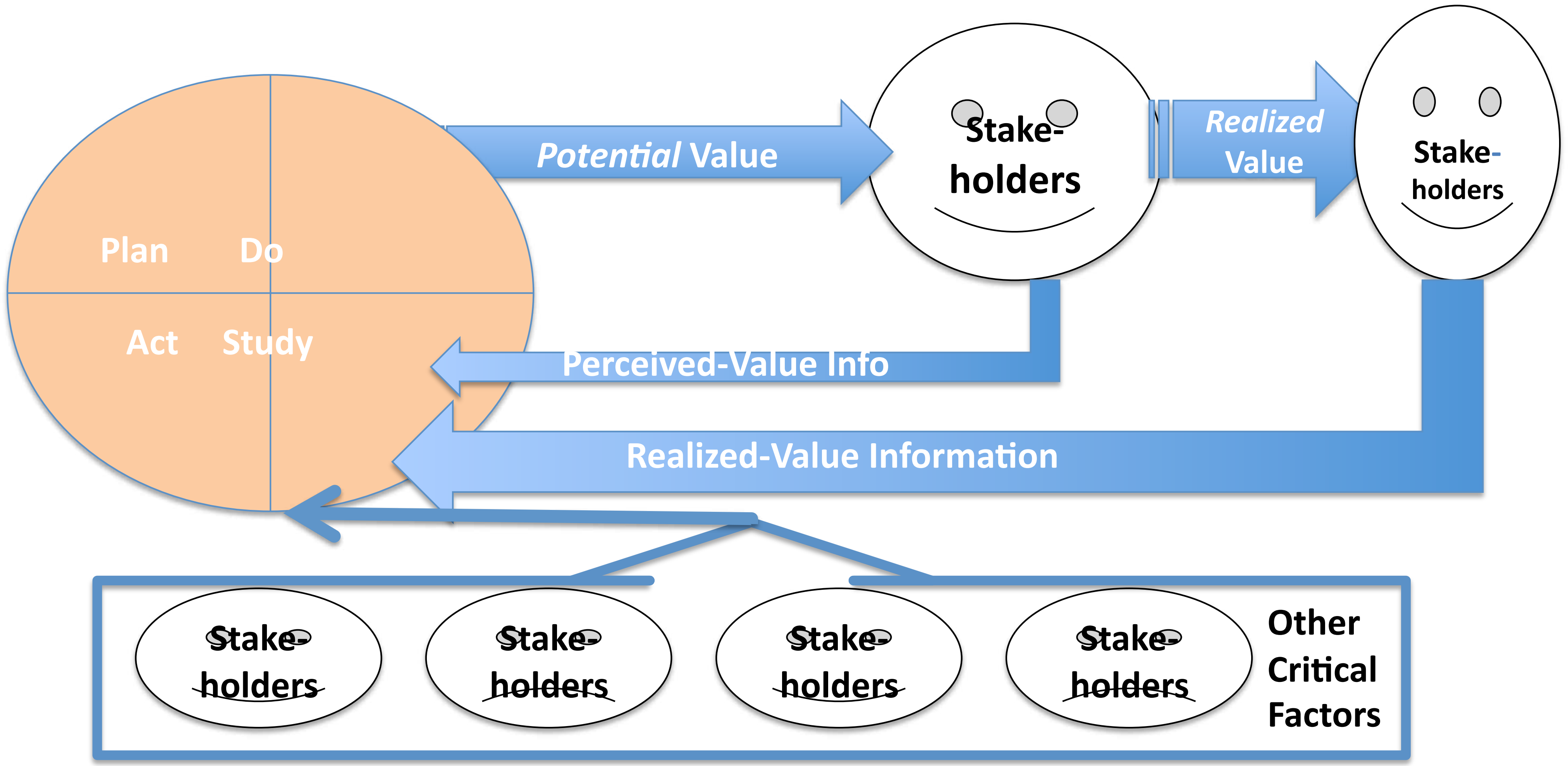
Long-Term *Real* Value Feedback

“This is the real value we have gotten to date, <u>and</u> 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.

The Simplest and Best Agile Project Method; ‘Evo’!

- Process Description
  - 1. Gather from all the key stakeholders the top few (5 to 20) most critical goals that the project needs to deliver.
    - Give each goal a reference name (a tag).
  - 2. For each goal, define a scale of measure and a ‘final’ goal level.
    - For example: *Reliable: Scale: Mean Time Before Failure, Goal: 1 month.*
  - 3. Define approximately 4 budgets for your most limited resources
    - (for example, time, people, money, and equipment).
  - 4. Write up these plans for the goals and budgets
    - (*Try to ensure this is kept to only one page*).
  - 5. Negotiate with the key stakeholders to formally agree the goals and budgets.
  - 6. Plan to deliver some benefit
    - (that is, progress towards the goals)
    - in *weekly* (or shorter) increments (Evo steps).
  - 7. Implement the project in Evo steps.
    - Report to project sponsors after each Evo step (weekly, or shorter) with you best available estimates or measures, for each performance goal and each resource budget.
    - *On a single page*, summarize the *progress to date* towards achieving the goals and the costs incurred.
  - 8. When all Goals are reached: ‘Claim success and move on’
    - a.	Free remaining resources for more profitable ventures

Agile project Management; Evo Policy

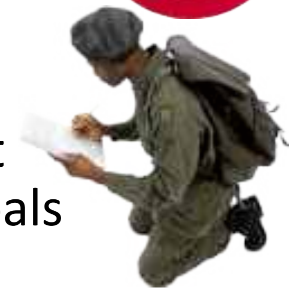
•! Policy

•! The project manager, and the project, will be judged exclusively on
   –! the relationship of progress towards achieving the goals
   –! versus the amounts of the budgets used.
   –! The project team will do anything legal and ethical to deliver the goal levels within the budgets.

•! The team will be paid and rewarded for
   –! benefits delivered
   –! in relation to cost.

•! The team will find their own work process and their own [illegible]sign.

•! As experience dictates, the team will be free to suggest to the project sponsors (stakeholders) adjustments to 'more realistic levels' of the goals and budgets.



QA 8: Quantify Maintainability Requirements:

- !Long term thinking
- !about maintenance and change capabilities:
- !avoid short sightedness.

A ‘Bad’ Requirement
“Rock solid robustness”

•! *While robustness is an essential HORROR requirement in all its uses, it is especially critical in MINING applications where the much longer job durations afford software defects (e.g. memory leaks) a greatly expanded opportunity to surface.*

•! In this regard,

•HORROR will provide the following features or attributes:

–! *Minimal down-time*

•! A critical HORROR objective is to have minimal downtime *<u>due to</u>* software failures.

•!This objective includes:

–! Mean time between forced restarts > 14 days

•! HORROR’s goal for mean time between forced restarts is greater than 14 days.

•! *Comment: This figure does not include restarts caused by hardware problems, e.g. poorly seated cards or communication hardware that locks up the system. MTBF for these items falls under the domain of the hardware groups.*

–! Restore system state < 10 minutes

•!Log scripts and test scripts, subsystem tests

–! *Built-in testability*

•! HORROR will provide the following features and attributes to facilitate testing.

–! Tool simulators

•! *GILB COMMENT:*

–! *For once a reasonable attempt was made to quantify the meaning of the requirement!*

–! *But is could be done much better*

–!

–! *As usual the set of designs to meet the requirement do not belong here.*

–!*And none of the designs make any assertion about how well (to what degree) they will meet the defined numeric requirements.*

–! *And as usual another guarantee of eternal costs in pursuit of a poorly defined requirement is most of the content.*

Real case of requirement for project costing over $100,000,000 without delivering testable results

Better Testable Definition of the Requirement:

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}.**

Defining One Component Clearly:

Software Downtime:

Type: Software Quality Requirement.

Ambition: ***to have minimal downtime due to softv***
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

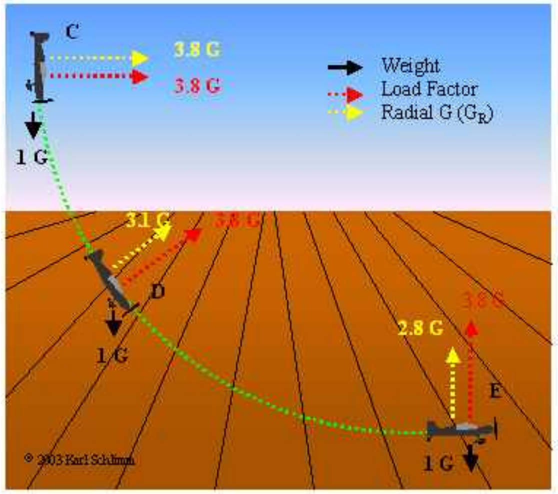
Defining a Second Component Clearly:

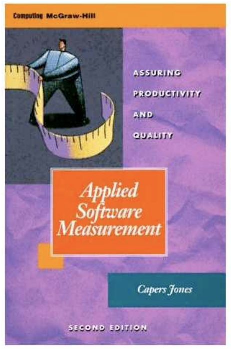
Restore Speed:
Type: Software Quality Requirement.
Ambition: Should an error occur (or the user otherwise desire to do so), Horizon shall be able to restore the system to a previously saved state in less than 10 minutes. <-6.1.2 HFA.

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.




Computing McGraw-Hill
Assuring Productivity and Quality
Applied Software Measurement
Capers Jones
Second Edition

System Lifetime Expectancy: Capers Jones: Think 18-25 Years

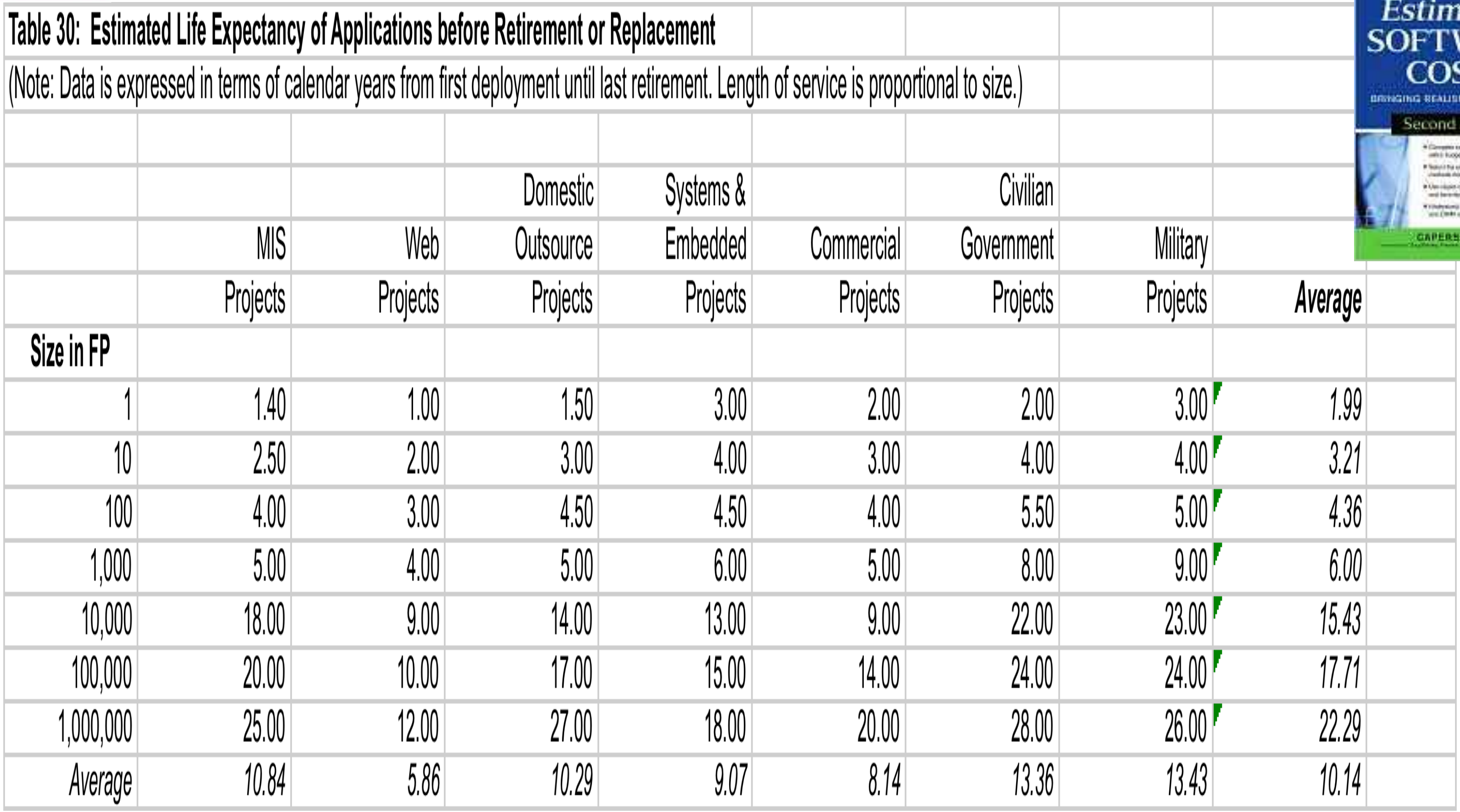


Table 30: Estimated Life Expectancy of Applications before Retirement or Replacement

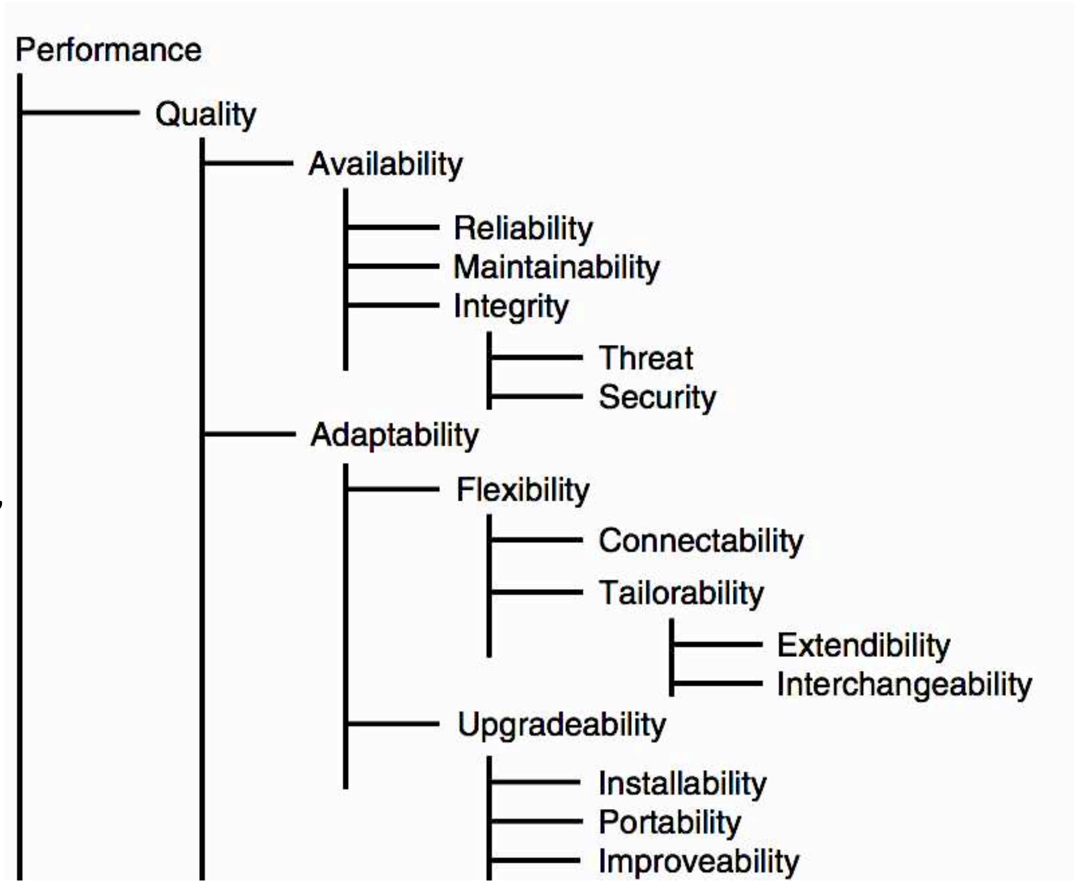
(Note: Data is expressed in terms of calendar years from first deployment until last retirement. Length of service is proportional to size.)

Size in FP	MIS Projects	Web Projects	Domestic Outsource Projects	Systems & Embedded Projects	Commercial Projects	Civilian Government Projects	Military Projects	Average
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

© Gilb.com 'Lean QA'

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.
-



The 'Maintainability' Breakdown into Sub-problems

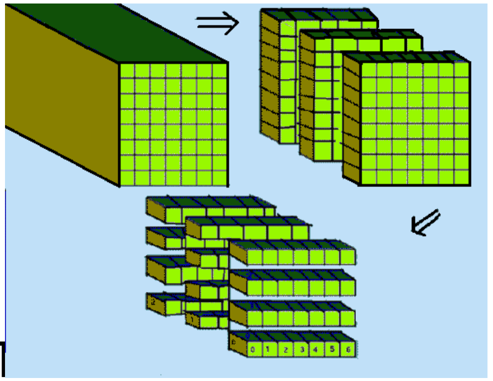
1. Problem Recognition Time.
   How can we reduce the time from bug actually occurs until it is recognized and reported?

2. Administrative Delay Time:
   How can we reduce the time from bug reported, until someone begins action on it?

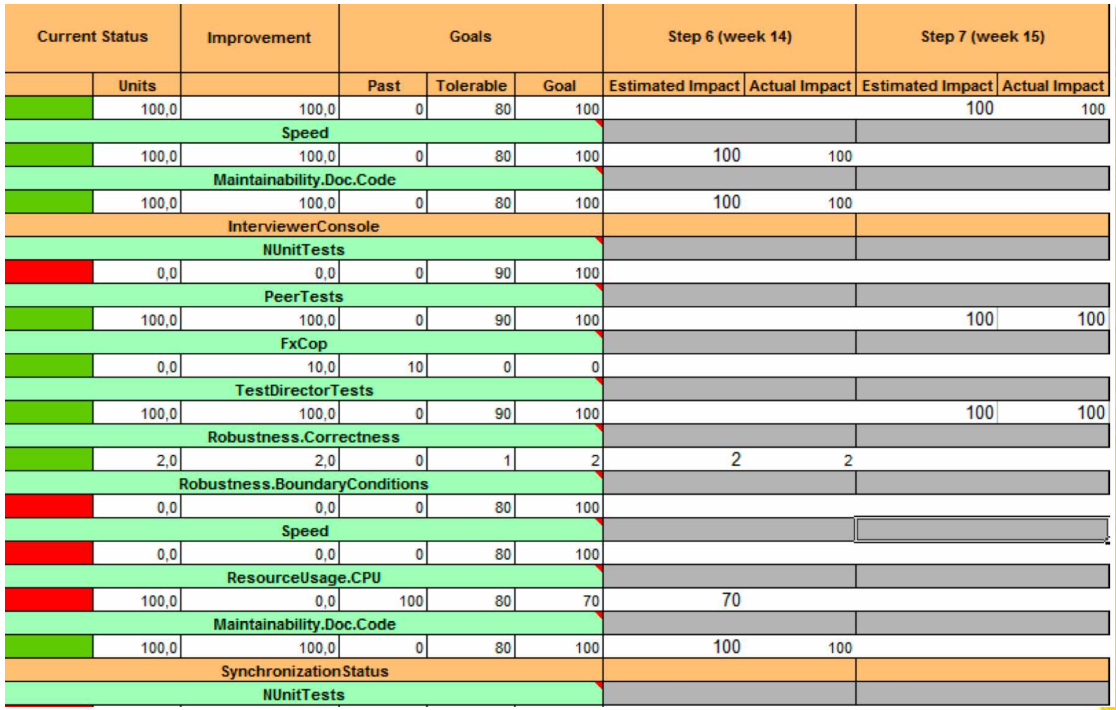
3. Tool Collection Time.
   How can we reduce the time delay to collect correct, complete and updated information to analyze the bug: source code, changes, database access, reports, similar reports, test cases, test outputs.

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 Ti
11. Customer Installation T
12. Customer Damage Analysis Tim
13. Customer Level Recovery Time
14. Customer QC of Recovery Time



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

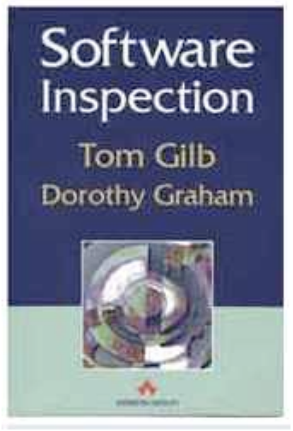
Robustness.Boundary Conditions

ResourceUsage.CPU

Maintainability.DocCode

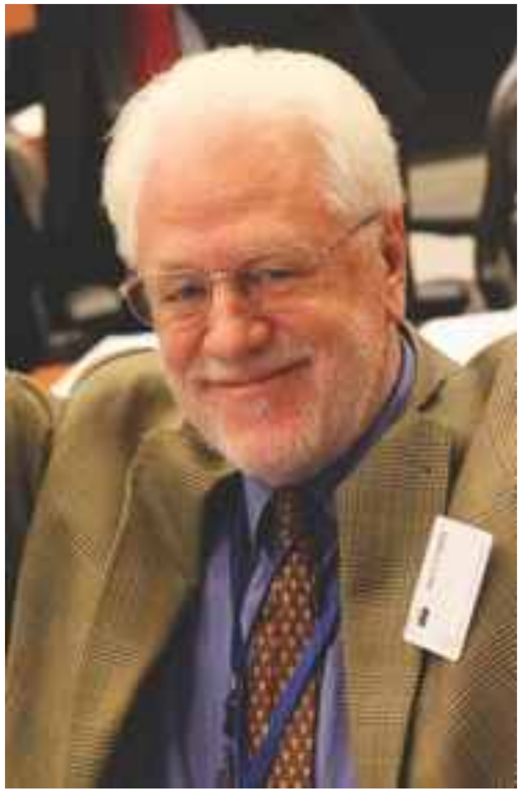
SynchronizationStatus

POT-SHOTS — Brilliant Thoughts in 17 words or less
SOMETHING'S WRONG WITH MY LIFE ~
SHOULD I TRY TO FIX IT, OR WAIT UNTIL I GET ANOTHER ?
© Ashleigh Brilliant
www.ashleighbrilliant.com

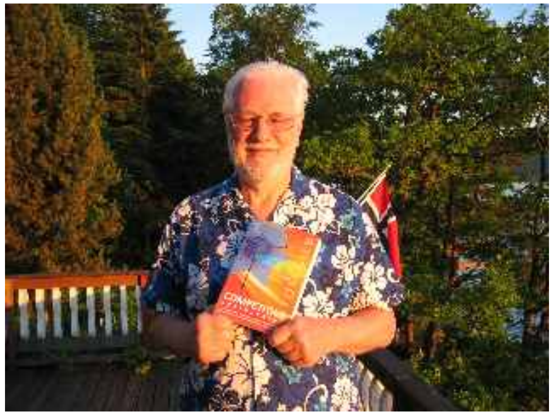


Last Slide!

- Questions: now, briefly
- After lecture, all during the conference
- By Email: tomsgilb at gmail.com
- @ imtomgilb



- Copy of these slides will be in Downloads: www.gilb.com
  - Library



Gilb: Biographical Data

- Tom Gilb (born 1940, California) has lived in UK since 1956, and Norway since 1958.
- He is the author of 9 published books, including “Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage”, 2005.
- He has taught and consulted world-wide for decades, including having direct corporate methods-change influence at major corporations such as Intel, HP, IBM, Nokia, Siemens, JP Morgan, Citigroup, Credit Suisse, Ericsson, Symbian.
- He has had documented his founding influence in Agile Culture, especially with the key common idea of *iterative development*.
- He coined the term 'Software Metrics' with his 1976 book of that title.
- He is co-author with Dorothy Graham of the static testing method book 'Software Inspection' (1993).
- He is known avoiding the oversimplified pop culture that regularly entices immature programmers to waste time and fail on their projects.
- More detail at www.Gilb.com

These QA Methods Are 'Lean'!

- Everything not adding value to the Customer is considered to be waste.
  - This includes:
    - unnecessary code and functionality
    - Delay in the software development process
    - Unclear requirements
    - Bureaucracy
    - Slow internal communication
  - Amplify Learning
    - The learning process is sped up by usage of short iteration cycles – each one coupled with refactoring and integration testing. Increasing feedback via short feedback sessions with Customers helps when determining the current phase of development and adjusting efforts for future improvements.
  - Decide as late as possible
  - Deliver as fast as possible
  - Empower the team
  - Build integrity in
    - separate components work well together as a whole with balance between flexibility, maintainability, efficiency, and responsiveness.
  - See the whole
    - "Think big, act small, fail fast; learn rapidly"

http://en.wikipedia.org/wiki/Lean_software_development