Super Methods: New Ways for Europe to do much better software ‘engineering’ than the rest of the world

one hour, + 15 minutes qa then a break, a separate qa session later

By Tom Gilb
TomsGilb@Gmail.com
www,Gilb.com
MASTER

Super Methods:
NEXT SLIDE = SUMMARY
New Ways for Europe to do much better software engineering than the rest of the world

- **Background:**
  - **Most of the world of software development is filled with poor practices, which have lead to decades-long consistent-failure to deliver expected software qualities, on time, and under budget.**
  - **The US-based latest fashion ideas are themselves poor conceptually, and then they are accepted and practiced uncritically, and in haphazard and second-rate ways**
  - **If Europe, as a high cost area, is to survive in the long term, it must now start to lead in delivering value for money. Old Europe is clearly hopeless and unmotivated. But, New Europe is capable and motivated: but maybe so young they have not yet learned about the powerful proven ideas that they can apply to master the software business.**
  - **The methods I am going to suggest are *well proven*, and well documented, in practice, with our own clients including IBM, HP, Citigroup, JP Morgan, Boeing, Ericsson, Sony and many smaller companies.**
  - **But, they are not well taught, and are not well known outside of the places that practice them. Those who choose to practice them can expect quick measurable improvements, and a competitive advantage over popular practices elsewhere.**
  - **In summary the methods represent a paradigm shift from software as a craft (coding) to software as a real engineering discipline. The engineers will beat, or manage, the 'Chinese army' of coders.**

Super Methods:
New Ways for Europe to do much better software engineering than the rest of the world

<

- Bad Practices = Project Failure
- US led culture is NOT impressive (where is EU ?)
- Can New Euope LEAD?
- These are Well Proven Methods
- NOT TAUGHT, NOT KNOWN
- Quick proven measurable results = NORMAL here
- a paradigm shift
  - from software as a craft (coding)
  - to software as a real engineering discipline.

Super Methods:
New Ways for us to do much better software engineering than others

- **The Super Methods**
  - Quality Quantification
  - Dynamic Design to Cost
  - Iteration and Incrementation with Multidimensional Quantified Quality and Cost Process Control
  - Defect Prevention Process (DPP)
  - *Systems* Engineering (for software projects)
  - 'No Cure, No Pay' Contracting, and Project Management
  - Specification Quality Control: SQC
  - Quantified Software *Process* Control: Numeric XE
  - Life Cycle Engineering of System and Product Adaptability

Super Methods:
New Ways for Europe to do much better software engineering than the rest of the world

- **The Super Methods**

Quality Quantification

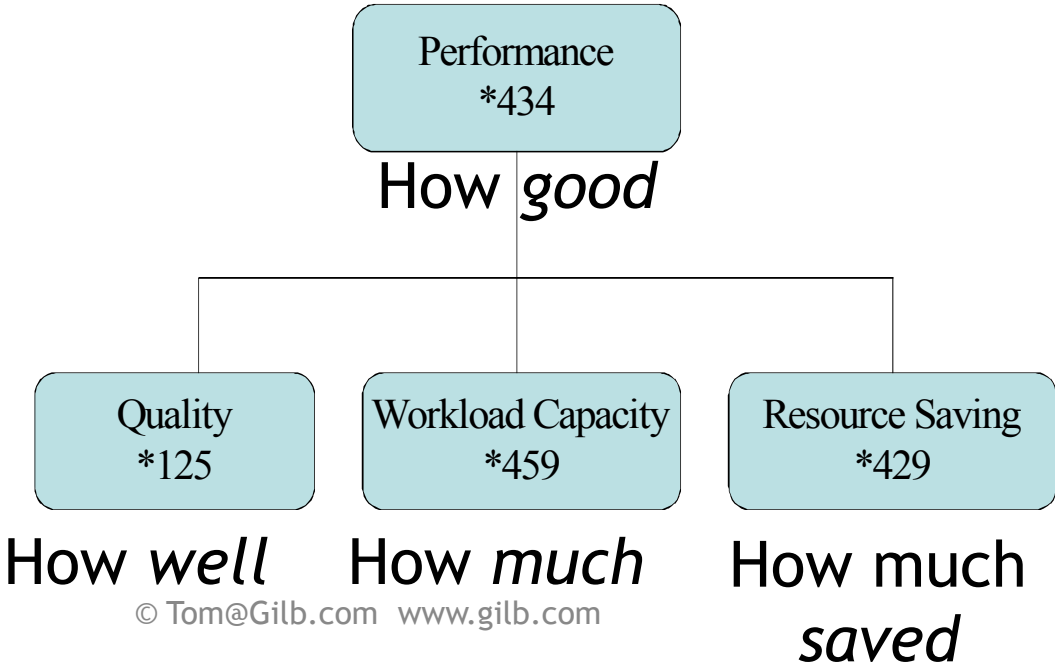
Putting 12345's on the '-ilities'

Quality: the concept, the noun

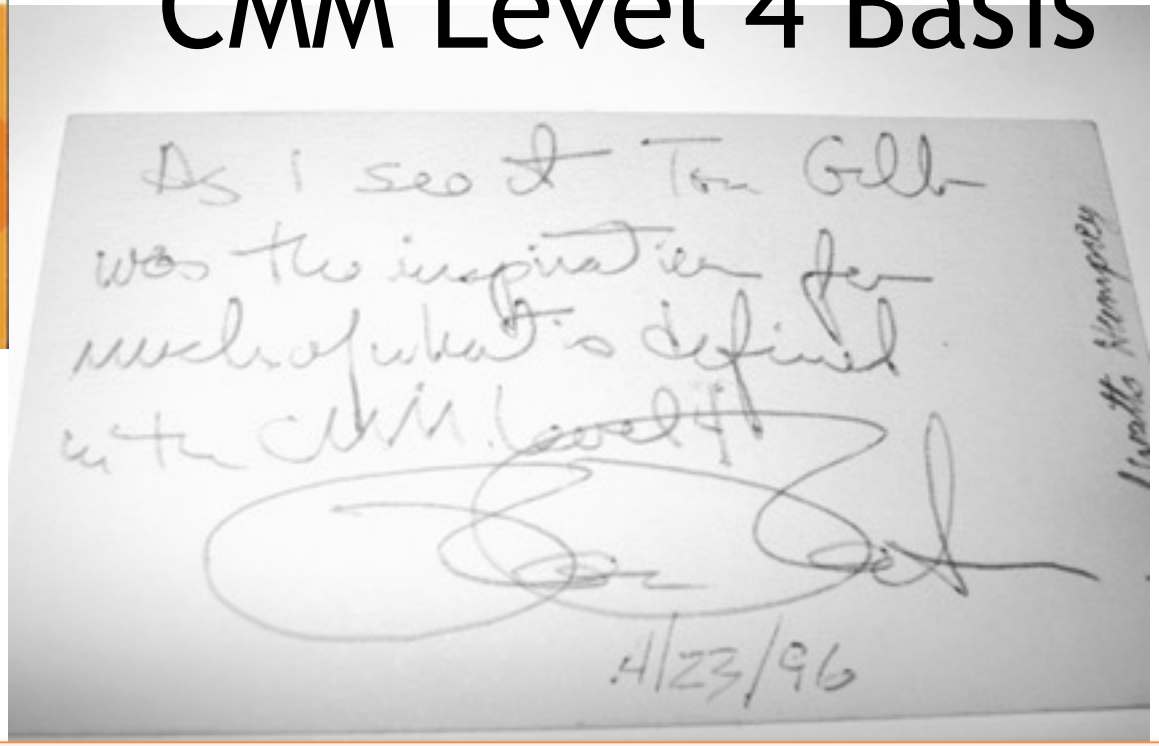
Planguage Concept *125, Version: March 20, 2003

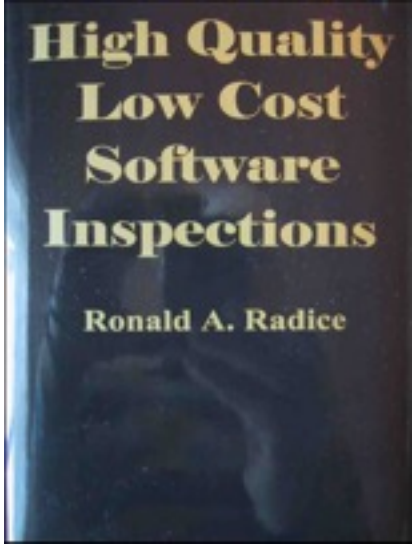
A '**quality**' is

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



CMM Level 4 Basis





- **"As I see it Tom Gilb was the inspiration for much of what is defined in CMM Level 4."**
- *Ron Radice (CMM Inventor at IBM, SEI Process Director) 1996 Salt lake City*
- *(agreed orally by Watts Humpreys - his IBM Boss)*
- *stt@stt.com www.stt.com*

Quality *is characterized by these traits*

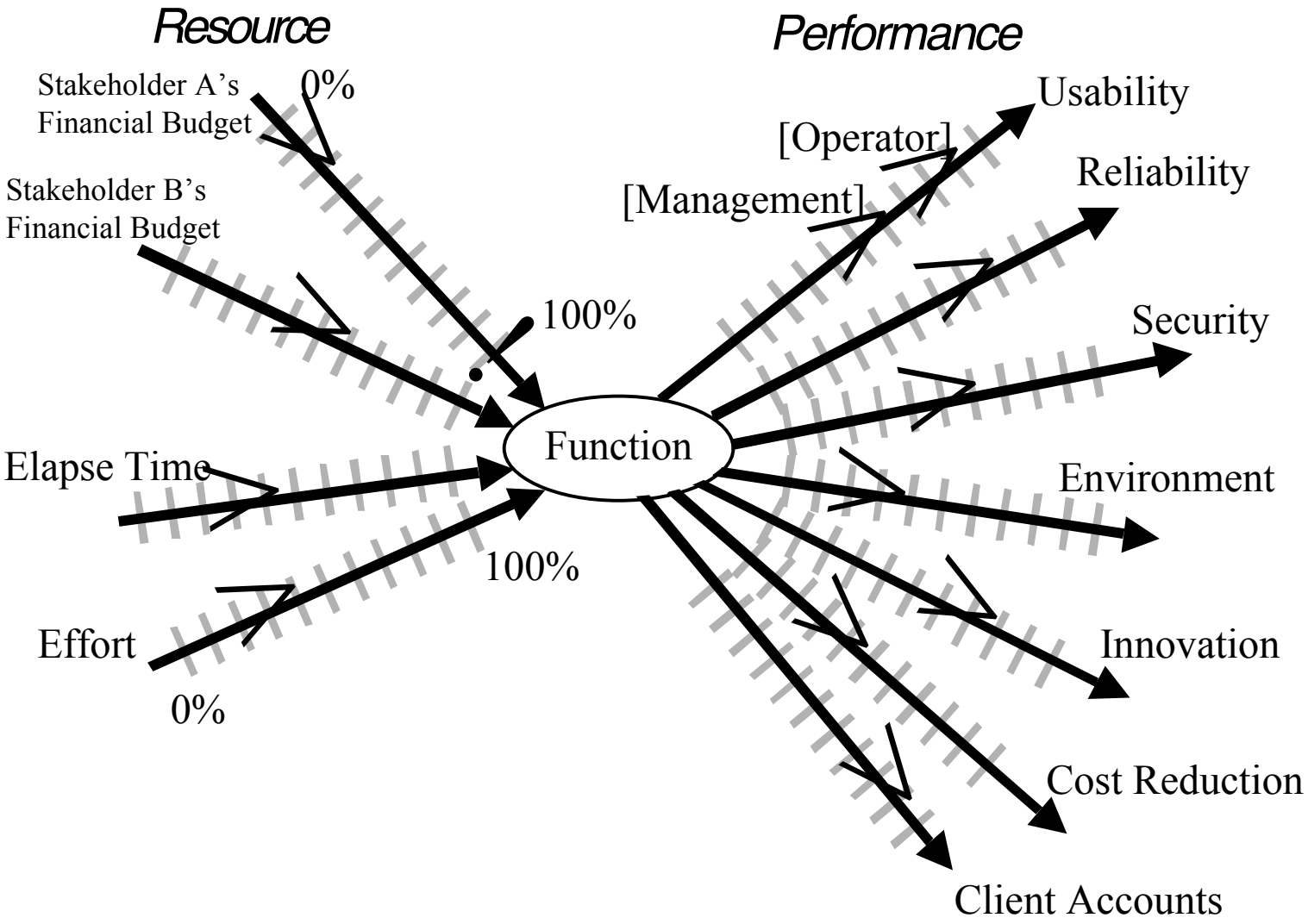
1. **Quality describes 'how well' a function is done.**
2. **Quality describes the partial effectiveness of a function (as do all other performance attributes).**
3. **Quality is valued to some degree by some stakeholders of the system**
4. **More quality is generally valued by stakeholders; especially if the increase is free, or lower cost, than the value of the increase.**
5. **Quality attributes can be articulated independently of the particular means (designs) used for reaching a specific quality level –**
6. **even though all quality levels depend on the particular designs used to achieve them.**
7. **A particular quality can be a described in terms of a complex concept, consisting of multiple elementary quality concepts.**
8. **Quality is variable (along a definable scale of measure: as are all scalar attributes).**
9. **Quality levels are capable of being specified quantitatively (as are all scalar attributes).**
10. **Quality levels can be measured in practice.**
11. **Quality levels can be traded off to some degree; with other system attributes valued more by stakeholders.**
12. **Quality can never be perfect (100%), in the real world.**
13. **There are some levels of a particular quality that may be outside the state of the art; at a defined time and circumstance.**
14. **When quality levels increase towards perfection, the resources needed to support those levels tend towards infinity.**

Quality *is characterized by these traits*

1. **Quality describes 'how well' a function is done.**
2. **Quality describes the partial effectiveness of a function (as do all other performance attributes).**
3. **Quality is valued to some degree by some stakeholders of the system**
4. **More quality is generally valued by stakeholders; especially if the increase is free, or lower cost, than the value of the increase.**
5. **Quality attributes can be articulated independently of the particular means (designs) used for reaching a specific quality level –**
6. **even though all quality levels depend on the particular designs used to achieve them.**
7. **A particular quality can be a described in terms of a complex concept, consisting of multiple elementary quality concepts.**
8. **Quality is variable (along a definable scale of measure: as are all scalar attributes).**
9. **Quality levels are capable of being specified quantitatively (as are all scalar attributes).**
10. **Quality levels can be measured in practice.**
11. **Quality levels can be traded off to some degree; with other system attributes valued more by stakeholders.**
12. **Quality can never be perfect (100%), in the real world.**
13. **There are some levels of a particular quality that may be outside the state of the art; at a defined time and circumstance.**
14. **When quality levels increase towards perfection, the resources needed to support those levels tend towards infinity.**

You can and should quantify Software and system qualities

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



What can we do *better* (or 'at all'), if we **quantify** quality ideas?

- **Evaluation** solutions/designs/architectures against the quantified quality requirements (Impact Estimation)
- **Test** and measure the degree to which solutions meet quality and cost expectations (when they were chosen)
- **Measure** evolutionary project **progress** towards quality goals
  - And get early & continuous improved estimates for time to completion
- **Communicate** quality goals much **better** to all parties (users, customers, developers, testers, lawyers)
- **Contract** for results
  - Pay for results only (not effort expended)
- **Reward** teams for results achieved
- **Motivate** technical people to focus on real business results
- **Simplify** requirements (the top few quantified- everything else is design)
- **Collect** numeric **data** about designs, processes, organizational structures, to learn and use in future.
- Permits systematic corporate or academic **research** of a development environment

What can we do *better* (or 'at all'), if we **quantify** quality ideas?

- **Evaluation** solutions/designs/architectures against the quantified quality requirements (Impact Estimation)
- **Test** and measure the degree to which solutions meet quality and cost expectations (when they were chosen)
- **Measure** evolutionary project **progress** towards quality goals
  - And get early & continuous improved estimates for time to completion
- **Communicate** quality goals much **better** to all parties (users, customers, developers, testers, lawyers)
- **Contract** for results
  - Pay for results only (not effort expended)
- **Reward** teams for results achieved
- **Motivate** technical people to focus on real business results
- **Simplify** requirements (the top few quantified- everything else is design)
- **Collect** numeric **data** about designs, processes, organizational structures, to learn and use in future
- Permits systematic corporate or academic **research** of a development environment

All Software and System Development and Maintenance Processes are dramatically affected by Quality Quantification

Lack of clear top level project objectives has seen real projects fail for $100+ million: personal experience, real case

Bad Objectives, for 8 years

*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 partial rewrite in example at right)*

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

Quantified Objectives (in Planguage),

Robustness.Testability:

Type: Software Quality Requirement.
Version: 20 Oct 2006-10-20
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.

VALUE CLARITY:
Quantify the most-critical project objectives on day 1

P&L-Consistency&T P&L: **Scale:** total adjustments btw Flash/Predict and Actual (T+1) signed off P&L. per day. **Past 60 Goal**: **15**

Speed-To-Deliver: **Scale**: average Calendar days needed from New Idea Approved until Idea Operational, for given Tasks, on given Markets.
Past [2009, Market = EURex, Task =Bond Execution] **2-3 months** ?
Goal [Deadline =End 20xz, Market = EURex, Task =Bond Execution] **5 days**

Operational-Control: **Scale**: % of trades per day, where the calculated economic difference between OUR CO and Marketplace/Clients, is less than "1 Yen"(or equivalent).
Past [April 20xx] **10%** change this to 90% NH **Goal** [Dec. 20xy] **100%**

Operational-Control.Consistent: **Scale**: % of defined [Trades] failing full STP across the transaction cycle. **Past** [April 20xx, Trades=Voice Trades] **95%**
Past [April 20xx, Trades=eTrades] **93%**
Goal [April 20xz, Trades=Voice Trades] <**95 ± 2%**>
Goal [April 20xz, Trades=eTrades] **98.5 ± 0.5 %**

Operational-Control.Timely.End&OvernightP&L Scale: number of times, per quarter, the P&L information is not delivered timely to the defined [Bach-Run].
Past [April 20xx, Batch-Run=Overnight] 1 **Goal** [Dec. 20xy, Batch-Run=Overnight] <**0.5**> **Past** [April 20xx, Batch-Run= **T+1**] **1 Goal** [Dec. 20xy, Batch-Run=End-Of-Day, Delay<1hour] **1**

Operational-Control.Timely.IntradayP&L Scale: number of times per day the intraday P&L process is delayed more than 0.5 sec.

Operational-Control.Timely.Trade-Bookings Scale: number of trades per day that are not booked on trade date. **Past** [April 20xx] **20 ?**

Front-Office-Trade-Management-Efficiency Scale: Time from Ticket Launch to trade updating real-time risk view
Past [20xx, Function = Risk Mgt, Region = Global] **~ 80s** +/- **45s** ??
Goal [End 20xz, Function = Risk Mgt, Region = Global] **~ 50%** better?
Managing Risk – Accurate – Consolidated – Real Time

Risk.Cross-Product Scale: % of financial products that risk metrics can be displayed in a single position blotter in a way appropriate for the trader (i.e. – around a benchmark vs. across the curve).
Past [April 20xx] **0%** 95%. **Goal** [Dec. 20xy] **100%**

Risk.Low-latency Scale: number of times per day the intraday risk metrics is delayed by more than 0.5 sec. **Past** [April 20xx, NA] **1% Past** [April 20xx, EMEA] ??% **Past** [April 20xx, AP] 100% **Goal** [Dec. 20xy] **0%**
Risk.Accuracy

Risk. user-configurable Scale: ??? pretty binary – feature is there or not – how do we represent?
Past [April 20xx] **1% Goal** [Dec. 20xy] **0%**

Operational Cost Efficiency Scale: <Increased efficiency (Straight through processing STP Rates)>

Cost-Per-Trade Scale: % reduction in Cost-Per-Trade
Goal (EOY 20xy, cost type = I 1 – REGION = ALL) **Reduce cost by 60%** (BW)
Goal (EOY 20xy, cost type = I 2 – REGION = ALL) **Reduce cost by x %**
Goal (EOY 20xy, cost type = E1 – REGION = ALL) **Reduce cost by x %**
Goal (EOY 20xy, cost type = E 2 – REGION = ALL) **Reduce cost by 100%**
Goal (EOY 20xy, cost type = E 3 – REGION = ALL) **Reduce cost by x %**

EVO Plan Confirmit 8.5 in **Evo Step Impact *Measurement***

4 product areas were attacked in all: **25 Qualities** concurrently, one quarter of a year. Total development staff = 13

9

8

3

3

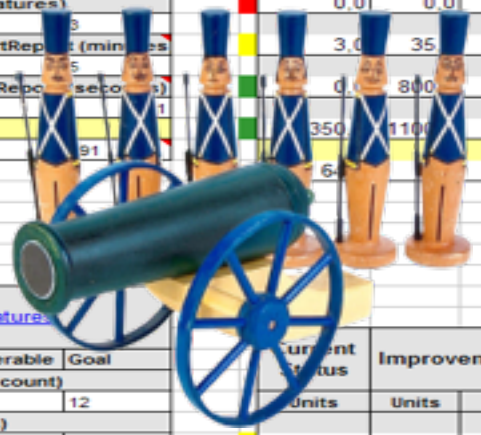
Impact Estimation Table: Reportal codename "Hyggen"

Current Status	Improvements		Reportal - E-SAT features		
Units	Units	%	Past	Tolerable	Goal
			Usability.Intuitivness (%)		
75,0	25,0	62,5	50	75	90
			Usability.Consistency.Visual (Elements)		
14,0	14,0	100,0	0	11	14
			Usability.Consistency.Interaction (Components)		
15,0	15,0	107,1	0	11	14
			Usability.Productivity (minutes)		
5,0	75,0	96,2	80	5	2
5,0	45,0	95,7	50	5	1
			Usability.Flexibility.OfflineReport.ExportFormats		
3,0	2,0	66,7	1	3	4
			Usability.Robustness (errors)		
1,0	22,0	95,7	7	1	0
			Usability.Replacability (nr of features)		
4,0	5,0	100,0	8	5	3
			Usability.ResponseTime.ExportRep (min		es
1,0	12,0	150,0	13	13	5
			Usability.ResponseTime.ViewRepo	seco	s)
1,0	14,0	100,0	15		1
			Development resources		
203,0			0		91

Current Status	Improvements		Survey Engine .NET		
Units	Units	%	Past	Tolerable	Goal
			Backwards.Compatibility (%)		
83,0	48,0	80,0	40	85	95
0,0	67,0	100,0	67	0	0
			Generate.WI.Time (small/medium/large seconds)		
4,0	59,0	100,0	63	8	4
10,0	397,0	100,0	407	100	10
94,0	2290,0	103,9	2384	500	180
			Testability (%)		
10,0	10,0	13,3	0	100	100
			Usability.Speed (seconds/user rating 1-10)		
774,0	507,0	51,7	1281	600	300
5,0	3,0	60,0	2	5	7
			Runtime.ResourceUsage.Memory		
0,0	0,0	0,0		?	?
			Runtime.ResourceUsage.CPU		
3,0	35	97,2	38	3	2
			Runtime.ResourceUsage.MemoryLeak		
0	800	100,0	800	0	0
			Runtime.Concurrency (number of users)		
350	1100	146,7	150	500	1000
			Development resources		
64			0		84

Current Status	Improvements		Reportal - MR Features		
Units	Units	%	Past	Tolerable	Goal
			Usability.Replacability (feature count)		
1,0	1,0	50,0	14	13	12
			Usability.Productivity (minutes)		
20,0	45,0	112,5	65	35	25
			Usability.ClientAcceptance (features count)		
4,4	4,4	36,7	0	4	12
			Development resources		
101,0			0		86

Current Status	Improvements		XML Web Services		
Units	Units	%	Past	Tolerable	Goal
			TransferDefinition.Usability.Efficiency		
7,0	9,0	81,8	16	10	5
17,0	8,0	53,3	25	15	10
			TransferDefinition.Usability.Response		
943,0	-186,0	######	170	60	30
			TransferDefinition.Usability.Intuitiveness		
5,0	10,0	95,2	15	7,5	4,5
			Development resources		
2,0			0		48



Evo's impact on Confirmit 9.0 product qualities Results from the second quarter of using Evo. 1/2

Product quality	Description	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%**

Product quality	Description	Customer value
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 ready for production. (Defined Survey: Complex survey, 60 questions, comprehensive JScripting.)	Time reduced by **83%** and error tracking increased by 25%

Evo's impact on Confirmit 9.0 product qualities Results from the second quarter of using Evo. 2/2

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%**

Code quality – ”green” week

- In these ”green” weeks, some of the deliverables will be less visible for the end users, but more visible for our QA department.
- We manage code quality through an Impact Estimation table

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

Speed

Maintainability

Nunit Tests

PeerTests

TestDirectorTests

Robustness.Correctness

Robustness.Boundary Conditions

ResourceUsage.CPU

Maintainability.DocCode

SynchronizationStatus

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

Exercise: Aspects of Love, or Love is a many splendored thing!

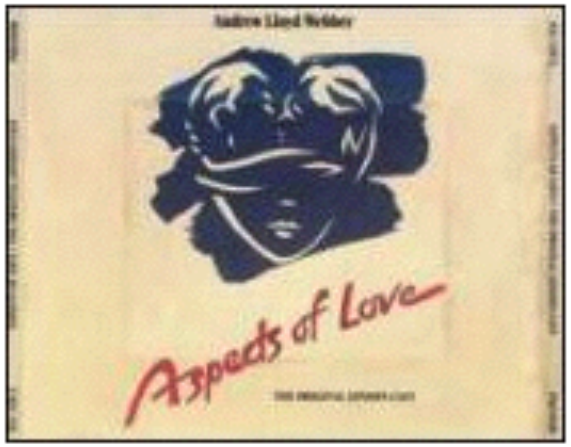
- Make a list of of love's many aspects
- Quantify a requirement for one of those aspects

See note for Sutra

Love Attributes:
Brainstormed By Dutch Engineers

- Kissed-ness
- Care
- Sharing
- Respect
- Comfort
- Friendship
- Sex
- Understandi
- Trust

- Support
- Attention
- Passion
- Satisfaction
- …
- …
- …



Trust [Caroline]

- **Love.Trust.Truthfulness**

  Ambition: No lies.

  Scale:

  Average Black lies/month from [defined sources].

  Meter:

  independent confidential log from sample of the defined sources.

  Past Lie Level:

  Past [My Old Mate, 2004] 42 <-Bart

  Goal

  [My Current Mate, Year = 2005] Past Lie Level/2

  Black: Defined: Non White Lies

- **Other aspects of Trust:**
  - **Broken Agreements**
  - **Late Appointments**
  - **Late delivery**
  - **Gossiping to Others**

Love: Biblical Dimensions : Bishop L Day, Boeing

The biblical citation (Book of First Corinthians I) **gives the quantification of the term "love" (agape in Greek).**

The 'quantification' for love would be as follows:

---------->

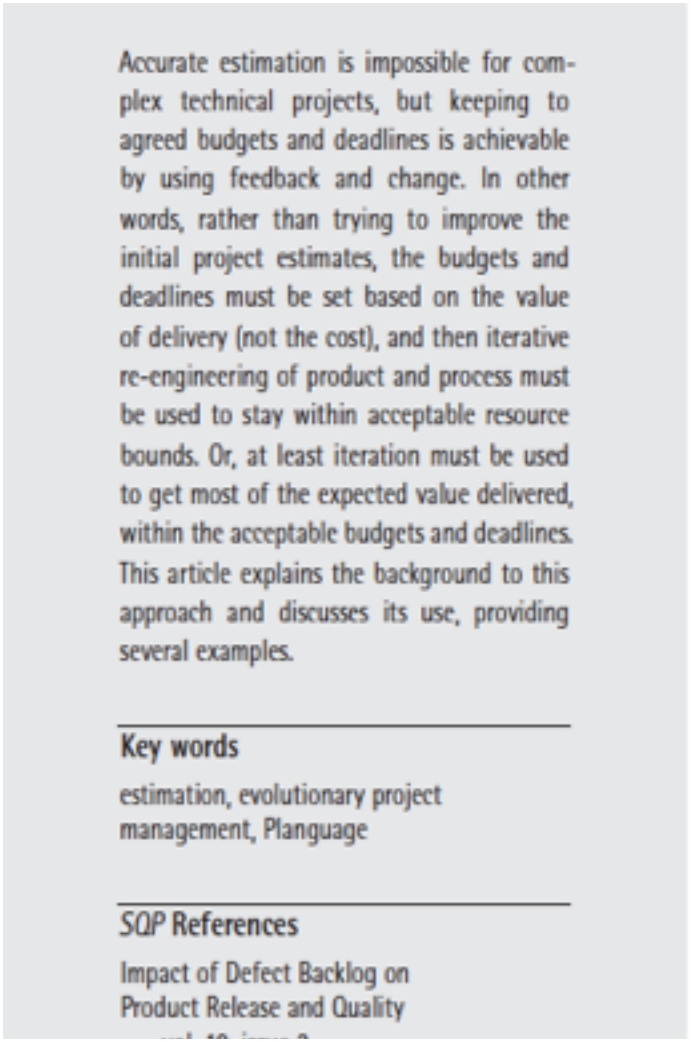
A person who loves acts the following way toward the person being loved:

1. **suffereth long**
2. **is kind**
3. **envieth not**
4. **vaunteth not itself, vaunteth...:**
   or, is not rash (Vaunt = extravagant self praise)
5. **is not puffed up**
6. **Doth not behave itself unseemly**
7. **seeketh not her own**
8. **is not easily provoked**
9. **thinketh no evil**
10. **Rejoiceth not in iniquity (=an unjust act)**
11. **rejoiceth in the truth**
12. **Beareth all things**
13. **believeth all things**
14. **hopeth all things**
15. **endureth all things**
16. **never faileth**

More Info Quality Quantification

- QQ Paper
  - **Quantifying Quality**: How to Tackle Quantification of the Critical Quality aspects for Projects for Both Requirements and Designs
  - http://www.gilb.com/tiki-download_file.php?fileId=124
- QQ Book Chapter (CE book)
  - http://www.gilb.com/tiki-download_file.php?fileId=26
- QQ Slides
  - http://www.gilb.com/tiki-download_file.php?fileId=131
- L. Day Love Quantification Paper
  - http://www.gilb.com/tiki-download_file.php?fileId=335
- Love Quantification Slides
  - Gilb, ACCU Lightening Talk
  - http://www.gilb.com/tiki-download_file.php?fileId=388

Dynamic Design to Cost



Accurate estimation is impossible for complex technical projects, but keeping to agreed budgets and deadlines is achievable by using feedback and change. In other words, rather than trying to improve the initial project estimates, the budgets and deadlines must be set based on the value of delivery (not the cost), and then iterative re-engineering of product and process must be used to stay within acceptable resource bounds. Or, at least iteration must be used to get most of the expected value delivered, within the acceptable budgets and deadlines. This article explains the background to this approach and discusses its use, providing several examples.

Key words

estimation, evolutionary project management, Planguage

SQP References

Impact of Defect Backlog on Product Release and Quality

Estimation: A Paradigm Shift Toward Dynamic Design-to-Cost and Radical Management

The Risk Principles

- **1. DRIVERS: If you have not specified all critical performance and quality levels numerically – you cannot estimate project resources for those vague requirements.**

- **2. EXPERIENCE: If you do not have experience data, about the resources needed for your technical solutions, then you cannot estimate the project resources.**

- **3. ARCHITECTURE: If you implement your project solutions *all at once*, without learning their costs and interactions incrementally – you cannot expect to be able to understand the results of many interactions.**

- **4. STAFF: If a complex and large professional project staff is an unknown set of people, or changes mid-project – you cannot expect to estimate the costs for so many human variables.**

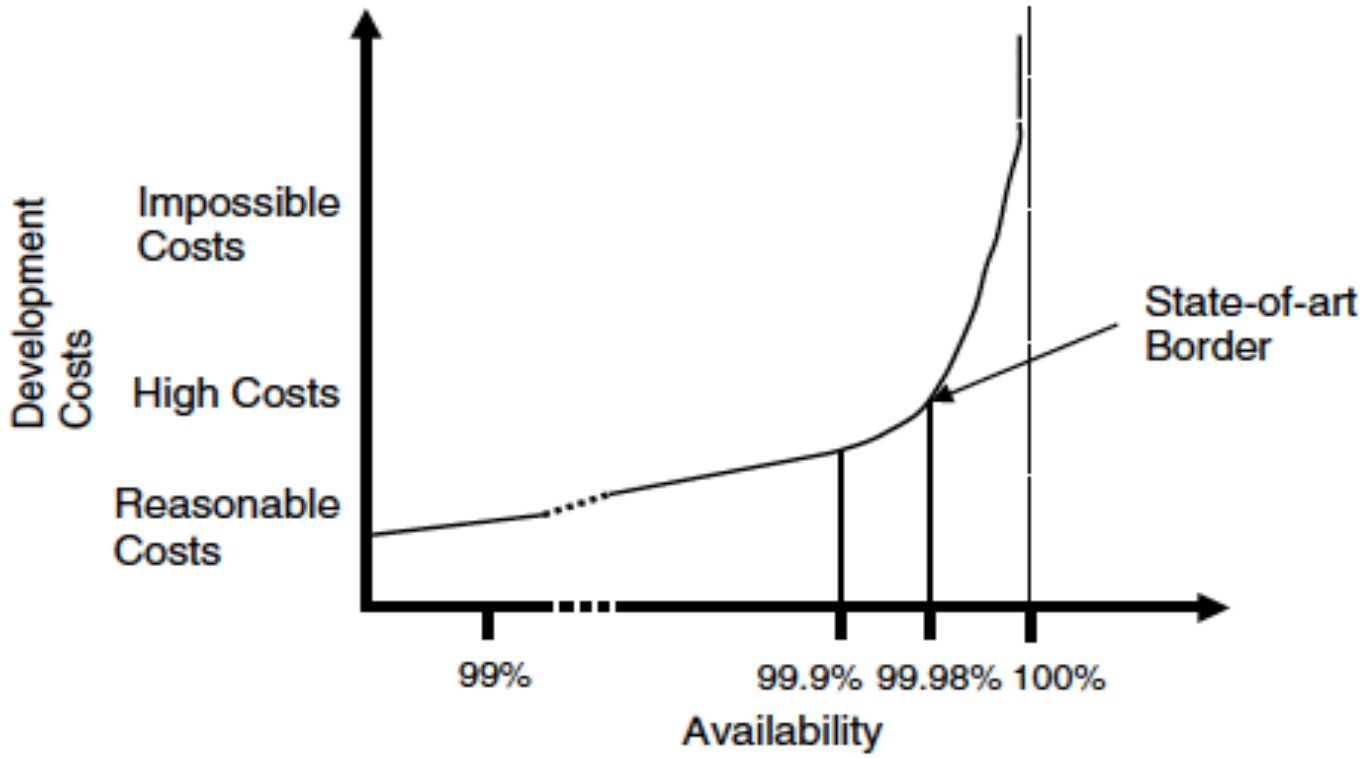
- **5. SENSITIVITY: If even the *slightest change* is made, after an 'accurate' estimation, to *any* of the requirements, designs or constraints – then the estimate might need to be changed *radically*. And – you probably will not have the information necessary to do it, nor the insight that you *need* to do it.**

The Risk Principles
Bottom Line =

- 1. DRIVERS: If you have not specified all critical performance and quality levels numerically - you cannot estimate project resources for those vague requirements.
- 2. EXPERIENCE: If you do not have experience data, about the resources needed for your technical solution, then you cannot estimate the project resources.
- 3. ARCHITECTURE: If you implement your project solutions *all at once*, without learning their costs and interactions incrementally - you cannot expect to be able to understand the results of many interactions.
- 4. STAFF: If a complex and large professional project staff is an unknown set of people, or changes mid-project - you cannot expect to estimate the costs for so many human variables.
- 5. SENSITIVITY: If even the *slightest change* is made, after an 'accurate' estimation, to *any* of the requirements, designs or constraints - then the estimate might need to be changed *radically*. And - you probably will not have the information necessary to do it, nor the insight that you *need* to do it.

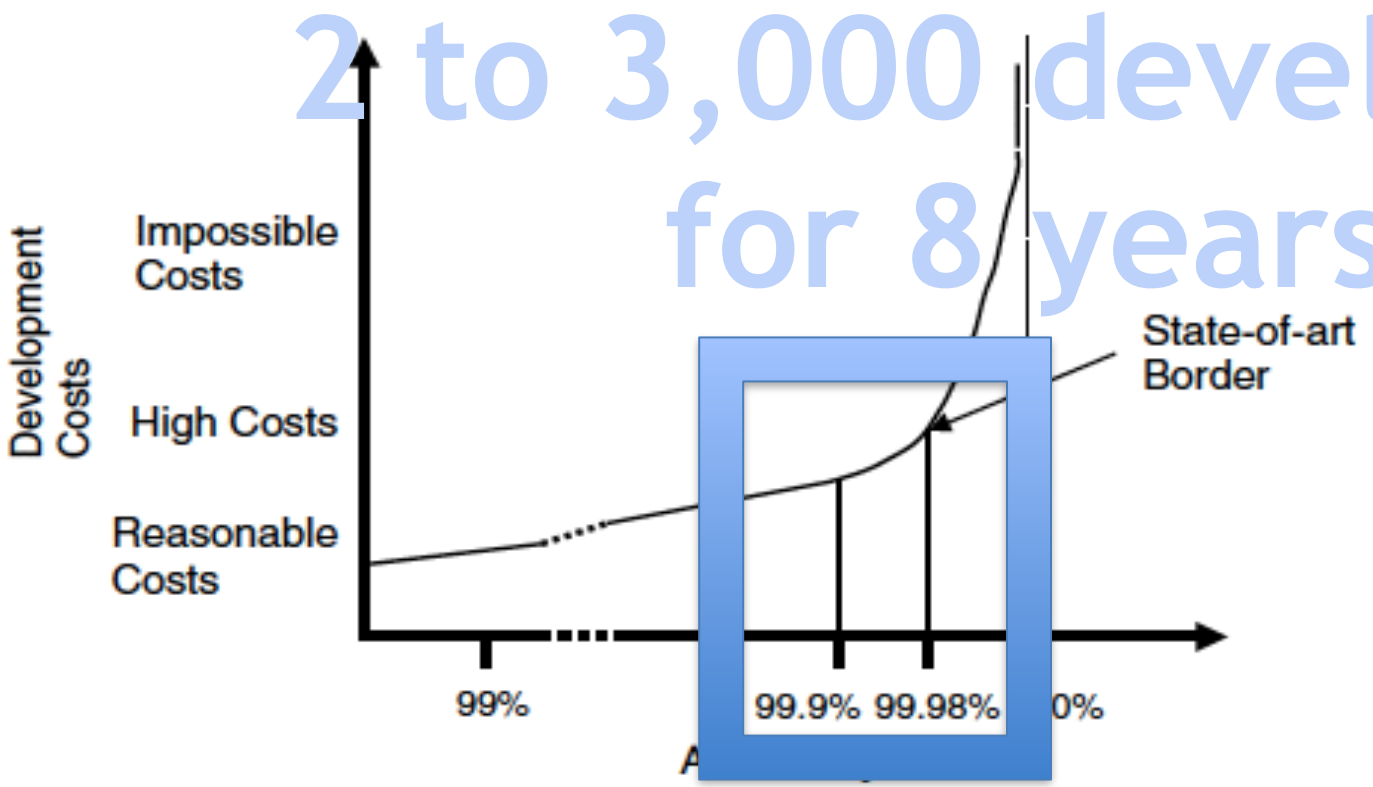
"You cannot hope To estimate with useful accuracy For large complex software projects

How much will ‘High Availability’ Cost? Just (99.90% to 99.98%) **00.08%** more?



See communication re.David Long, AT&T 2010 in pptx note here

How much will ‘High Availability’ Cost?



The Control Principles

(detailed in paper and slides referenced at end here)

6. LEARN SMALL: Carry out projects in *small increments* of delivering requirements – so you can measure *results* and *costs*, against (short term) estimates.

7. LEARN ROOT: If incremental costs for a given requirement level (and its designs) deviate negatively from estimates – analyze the root cause, and change anything about the next increments that you believe might get you back on track.

8. PRIORITIZE CRITICAL: You will have to *prioritize* your most critical requirements and constraints: there is no guarantee you can achieve them all. Deliver 'high-value for resources-used' *first*.

9. RISK FAST: You should probably implement the design ideas with the highest value, with regard to cost and risk, early.

10. APPLY NOW: Learn early, learn often, learn well; and apply the learning to your current project.

The Control Principles

(detailed in paper and slides referenced at end here)

6. LEARN SMALL: Carry out projects in *small increments* of delivering requirements - so you can measure *results* and *costs*, against (short term) estimates.

7. LEARN ROOT: If incremental costs for a given requirement level (and its designs) deviate negatively from estimates - analyze the root cause, and change anything about the next increments that you believe might get you back on track.

8. PRIORITIZE CRITICAL: You will have to *prioritize* your most critical requirements and constraints: there is no guarantee you can achieve them all. Deliver 'high-value for resources-used' *first*.

9. RISK FAST: You should probably implement the design ideas with the highest value, with regard to cost and risk, early.

10. APPLY NOW: Learn early, learn often, learn well; and apply the learning to your current project.

You *can* dynamically adjust Your project To meet budgets and deadlines

We can *simplify* The Control Principles

1. **Do something of value**
   in a short time

2. **Measure values and costs**

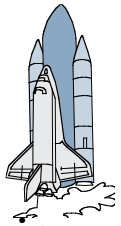
3. **Adjust what you do next,**
   if necessary

Repeat
until you no longer can find value for money

In the Cleanroom Method, developed by IBM's Harlan Mills (1980) they report (in Fact this is an Early 'Agile' Method!!)

- *"Software Engineering began to emerge in FSD" (IBM Federal Systems Division, from 1996 a part of Lockheed Martin Marietta) "some ten years ago [Ed. about 1970] in a continuing evolution that is still underway:*
- *Ten years ago general management **expected the worst** from software projects – cost overruns, late deliveries, unreliable and incomplete software*
- *Today [Ed. 1980!], **management has learned to expect on-time**, within budget, deliveries of high-quality software. A Navy helicopter ship system, called LAMPS, provides a recent example. LAMPS software was a four-year project of over 200 person-years of effort, developing over three million, and integrating over seven million words of program and data for eight different processors distributed between a helicopter and a ship in 45 incremental deliveries [Ed. Note 2%!]. Every one of those deliveries **was on time and under budget***

- *A more extended example can be found in the NASA space program,*
- *- Where in the past ten years, FSD has managed some 7,000 person-years of software development, developing and integrating over a hundred million bytes of program and data for ground and space processors in over a dozen projects.*



- *- There **were few late** or overrun deliveries in that decade, **and none at all** in the past four years."*

In the Cleanroom Method, developed by IBM's Harlan Mills (IBM SJ 4.1980) th
reported:
Simplified Slide

- *"Software Engineering at IBM Federal Systems Division*
- *Before 'Cleanroom' Method- cost overruns, late deliveries, unreliable and incomplete software*
- *Today [Ed. 1980!],* ***management has learned to expect on-time, within budget, deliveries of high-quality software.***
  - *A Navy helicopter ship system, called LAMPS ship in 45 incremental deliveries.*
  - ***Every one of those deliveries was on time and under budget***
- *A more extended example can be found in the* ***NASA space program,***
- *- There* ***were few late or overrun deliveries in that decade,***
- *and* ***none at all in the past four years.****"*

Recent (20 Sept, 2011) Report on Gilb Evo method (Richard Smith, Citigroup)

- http://rsbatechnology.co.uk/blog:8
- Back in 2004, I was employed by a large investment bank in their FX e-commerce IT department as a business analyst.
- The wider IT organisation used a complex waterfall-based project methodology that required use of an intranet application to manage and report progress.
- However, it's main failings were that it almost **totally missed the ability to track delivery of actual value improvements to a project's stakeholders**, and **the ability to react to changes in requirements and priority for the project's duration.**
- The toolset generated lots of charts and stats that provided **the illusion of risk control**. but actually provided very little help to the analysts, developers and testers actually doing the work at the coal face.
- The proof is in the pudding;
  - I have **used Evo** (albeit in disguise sometimes) on two large, high-risk projects in front-office investment banking businesses, and several smaller tasks.
  - On the largest critical project, the original business functions & performance objective **requirements document, which included no design, essentially remained unchanged** over the 14 months the project took to deliver,
  - but the detailed **designs** (of the GUI, business logic, performance characteristics) **changed many many times**, guided by lessons learnt and **feedback** gained by delivering a succession of early deliveries to real users.
  - In the end, the new system responsible for 10s of USD billions of notional risk, **successfully went live over over one weekend for 800 users worldwide**, and **was seen as a big success by the sponsoring stakeholders.**

" I attended a 3-day course with you and Kai whilst at Citigroup in 2006"

Advantages with Control Principles

1. You cannot waste much time or money before you realize that you have false ideas

2. You can deliver value early, and keep people happy

3. You are forced to think about the whole system, including people (not just code)

4. So you are destined to see the true costs of delivering value – not just the code costs

5. You will learn a general method that you can apply for the rest of your career.

Disadvantages of the Control Principles

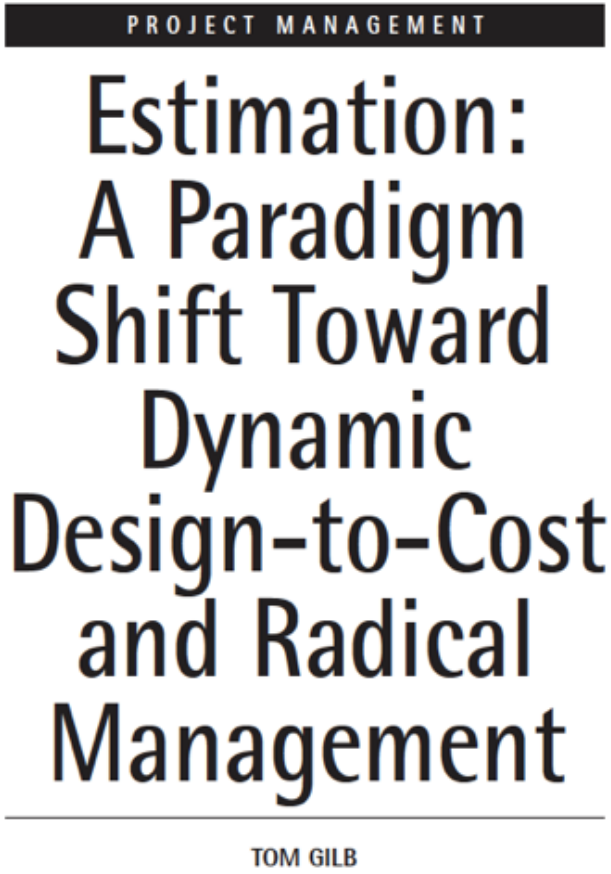
1. You cannot hide your ignorance from yourself any longer

2. You might have to do something not taught at school, or not taught in textbooks

3. There will always be people who criticize anything different or new

4. You cannot continue to hide your lack of ability to produce results, inside a multi-year delayed project.

Dynamic Design to Cost: more info



More Info

- Paper on Estimation with many links
  - Volume 13 Issue 2 of SQP journal - the March 2011 version.
  - http://www.gilb.com/tiki-download_file.php?fileId=460
  - Estimation: A Paradigm Shift Toward Dynamic Design-to Cost and Radical Management
  -
- Slides made for BCS SPA June 1 2011
  - 'Estimation, a Waste of Time'
  - http://www.gilb.com/tiki-download_file.php?fileId=470

Iteration and Incrementation with Multidimensional Quantified Quality and Cost Process Control

EVO Plan Confirmit 8.5 in **Evo Step Impact *Measurement***

4 product areas were attacked in all: **25 Qualities** concurrently, one quarter of a year. Total development staff = 13

9

8

3

3

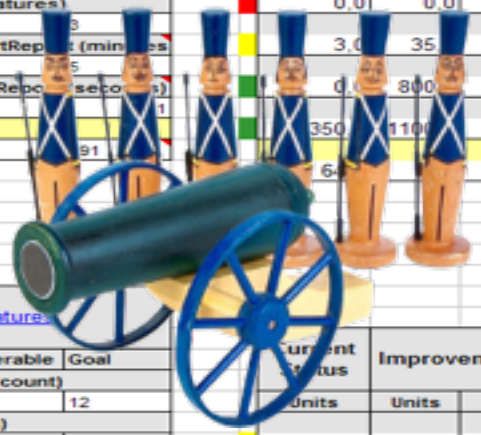
Impact Estimation Table: Reportal codename "Hyggen"

Current Status	Improvements		Reportal - E-SAT features		
Units	Units	%	Past	Tolerable	Goal
			Usability.Intuitivness (%)		
75,0	25,0	62,5	50	75	90
			Usability.Consistency.Visual (Elements)		
14,0	14,0	100,0	0	11	14
			Usability.Consistency.Interaction (Components)		
15,0	15,0	107,1	0	11	14
			Usability.Productivity (minutes)		
5,0	75,0	96,2	80	5	2
5,0	45,0	95,7	50	5	1
			Usability.Flexibility.OfflineReport.ExportFormats		
3,0	2,0	66,7	1	3	4
			Usability.Robustness (errors)		
1,0	22,0	95,7	7	1	0
			Usability.Replacability (nr of features)		
4,0	5,0	100,0	8	5	3
			Usability.ResponseTime.ExportRep (min		es
1,0	12,0	150,0	13	13	5
			Usability.ResponseTime.ViewRepo	seco	s)
1,0	14,0	100,0	15		1
			Development resources		
203,0			0		91

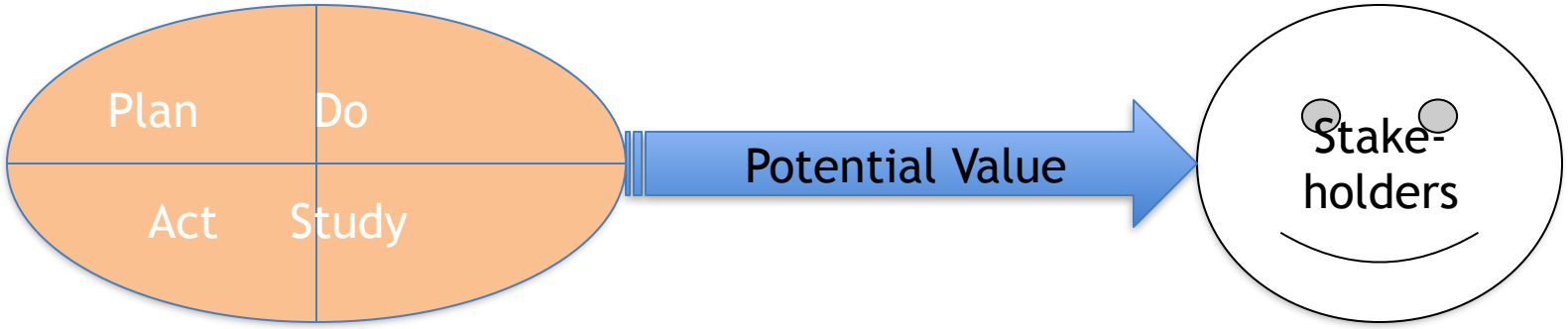
Current Status	Improvements		Survey Engine .NET		
Units	Units	%	Past	Tolerable	Goal
			Backwards.Compatibility (%)		
83,0	48,0	80,0	40	85	95
0,0	67,0	100,0	67	0	0
			Generate.WI.Time (small/medium/large seconds)		
4,0	59,0	100,0	63	8	4
10,0	397,0	100,0	407	100	10
94,0	2290,0	103,9	2384	500	180
			Testability (%)		
10,0	10,0	13,3	0	100	100
			Usability.Speed (seconds/user rating 1-10)		
774,0	507,0	51,7	1281	600	300
5,0	3,0	60,0	2	5	7
			Runtime.ResourceUsage.Memory		
0,0	0,0	0,0		?	?
			Runtime.ResourceUsage.CPU		
3,0	35	97,2	38	3	2
			Runtime.ResourceUsage.MemoryLeak		
0	800	100,0	800	0	0
			Runtime.Concurrency (number of users)		
350	1100	146,7	150	500	1000
			Development resources		
64			0		84

Current Status	Improvements		Reportal - MR Features		
Units	Units	%	Past	Tolerable	Goal
			Usability.Replacability (feature count)		
1,0	1,0	50,0	14	13	12
			Usability.Productivity (minutes)		
20,0	45,0	112,5	65	35	25
			Usability.ClientAcceptance (features count)		
4,4	4,4	36,7	0	4	12
			Development resources		
101,0			0		86

Current Status	Improvements		XML Web Services		
Units	Units	%	Past	Tolerable	Goal
			TransferDefinition.Usability.Efficiency		
7,0	9,0	81,8	16	10	5
17,0	8,0	53,3	25	15	10
			TransferDefinition.Usability.Response		
943,0	-186,0	######	170	60	30
			TransferDefinition.Usability.Intuitiveness		
5,0	10,0	95,2	15	7,5	4,5
			Development resources		
2,0			0		48



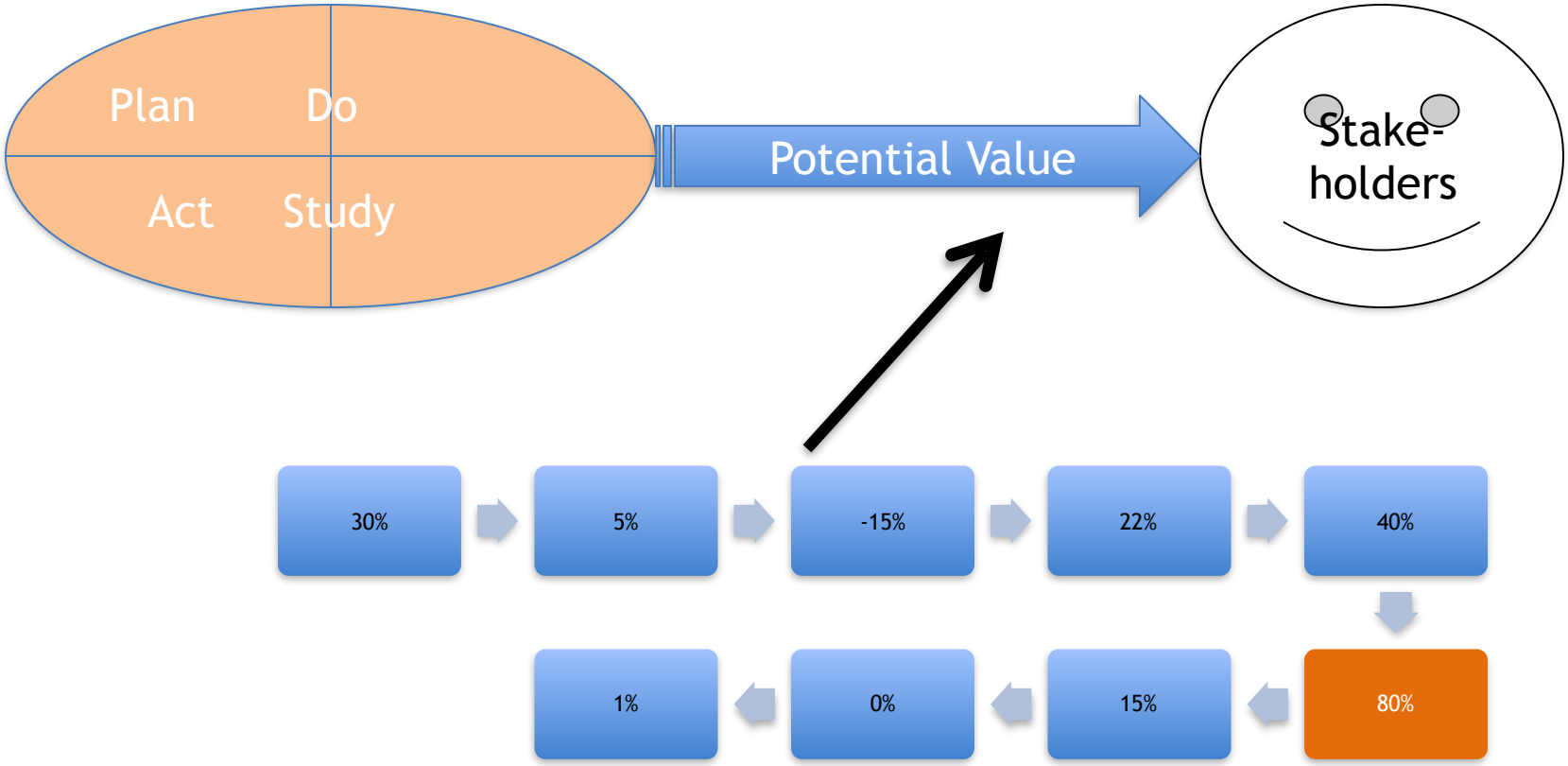
Primary Evo Concept:
Deliver *Potential* Value



The Evo Cycle:
Viewed as a Deming PDSA Cycle

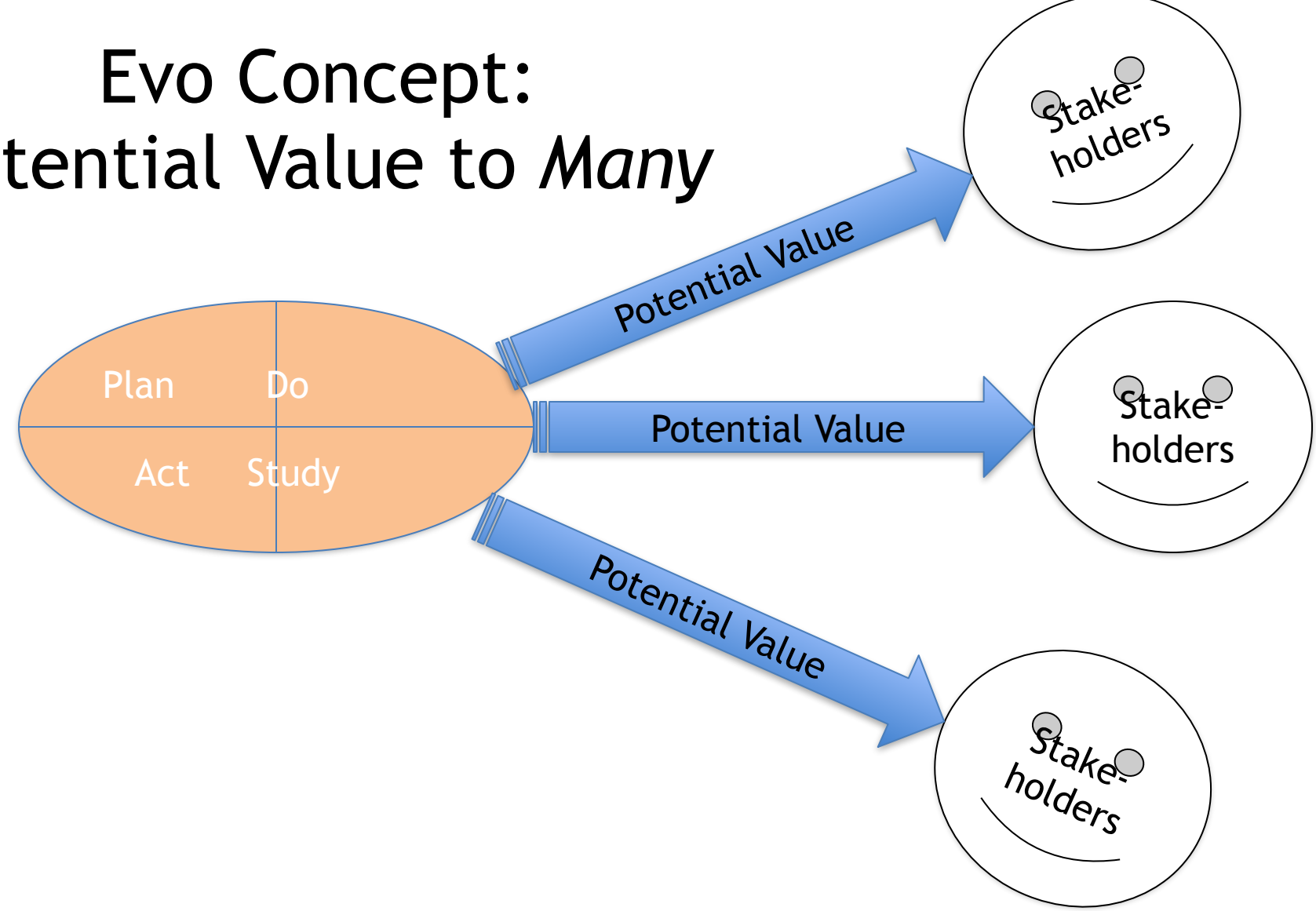
- Incremental Value Delivery to Stakeholders

Deliver the highest value for resources



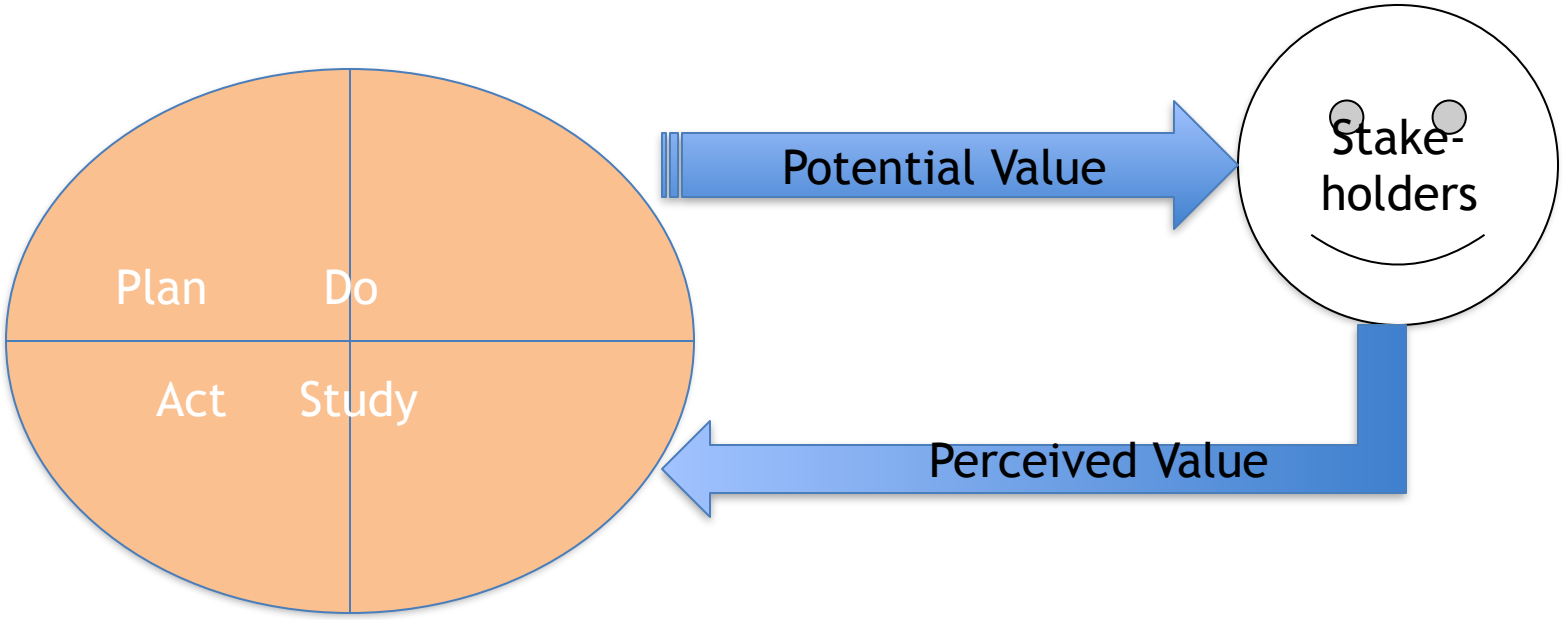
HIGHEST AVAILABLE Incremental Value Delivery to Stakeholders

Evo Concept: Potential Value to *Many*



- Incremental Value Deliveries to *Many* Stakeholders

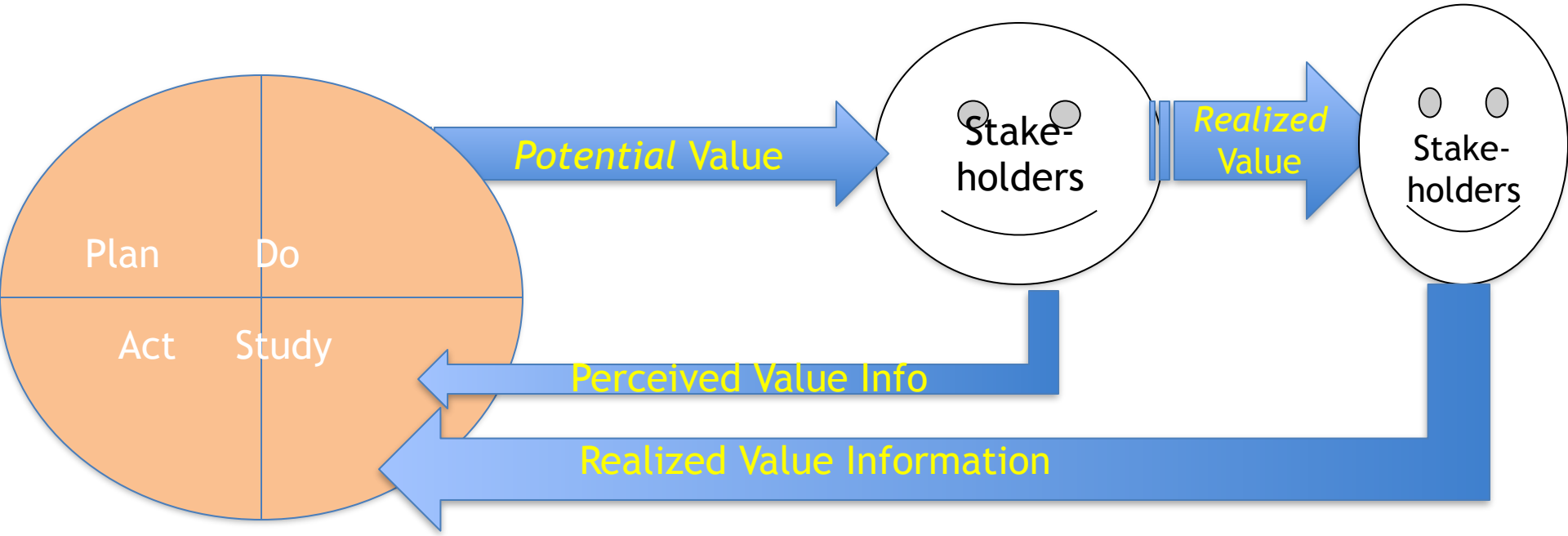
Evo Concept: Short Term Feedback
"This looks like a change I can get value from!"



- Initial Feedback from Stakeholders, after Evo Cycle delivery

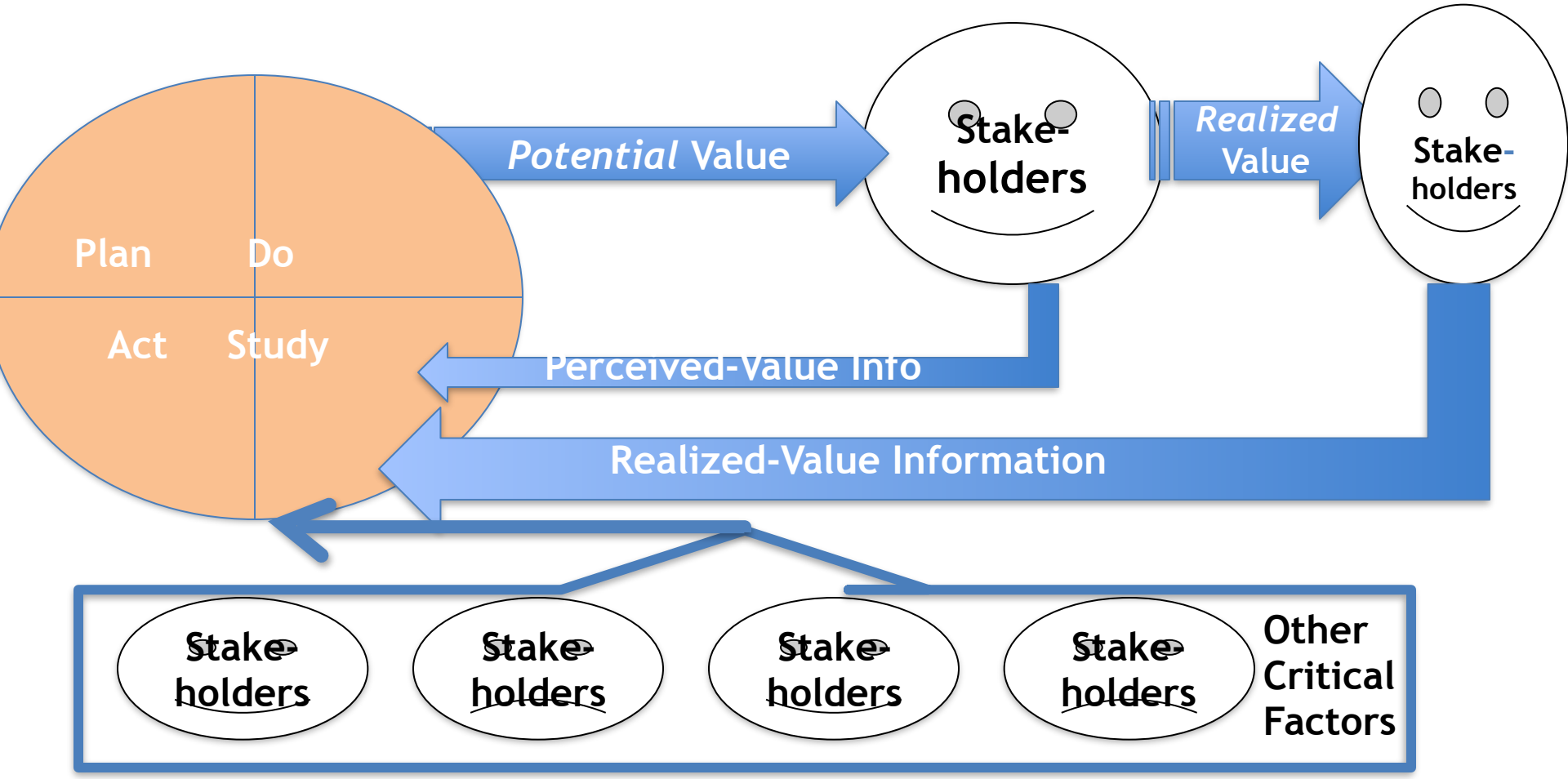
Long-Term *Real* Value Feedback

"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.

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 design
- 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.

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 design
- 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.

Iteration and Incrementation with Multidimensional Quantified Quality and Cost Process Control

- **REFERENCES**

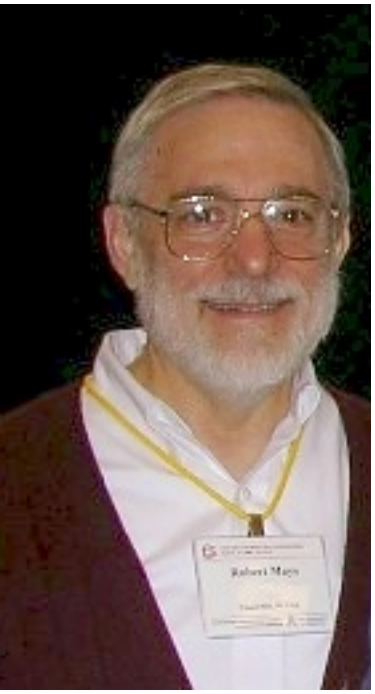
  1. Book "Competitive Engineering" (ask me for full digital copy by email)
  - **Chapter 10 Evo**
  - **http://www.gilb.com/tiki-download_file.php?fileId=77**
  -

- **2. Book: Kai Gilb, Evo:**
- **http://www.gilb.com/tiki-download_file.php?fileId=27**
-

- **Papers:**
- Confirmit Case of Evo
  - How we rapidly created faster, more user-friendly, and more productive software products for a competitive multi-national market.
  - http://www.gilb.com/tiki-download_file.php?fileId=32

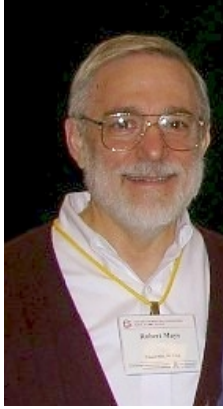
- Evo Principles
  - http://www.gilb.com/tiki-download_file.php?fileId=59
- What are Evo methods?
  - http://www.gilb.com/tiki-download_file.php?fileId=55
- **SLIDES**
  - **Evo Method with Metrics**
  - http://www.gilb.com/tiki-download_file.php?fileId=150

Defect Prevention Process (DPP) = CMM(i) 5



- Robert G. Mays and Linda Jones IBM

How does DPP work?



DPP Detail

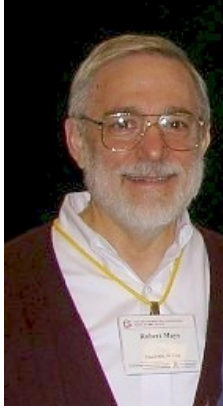
- **After an Inspection Process, Major defects are examined by the checking team. Half an hour sessions.**
  - **They are looking at a colleagues work, colleague is there (the source of defects: knows why)**
- **They arbitrarily select one, of a small group of recurrent types of defects, to work on (3 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.**

DPP Process Summary

- **Grass roots teams quickly analyze frequent defects causes**
- **Brainstorm root causes**
- **Brainstorm cures for root causes**
- **Pilot the changes locally**

- **If successful**
  - **Spread corporate wide**

Effects of DPP: Grass roots wisdom



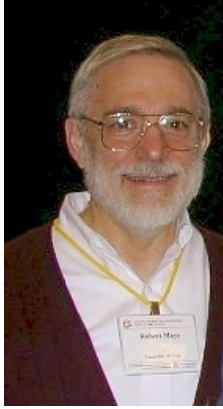
Detail

- **Systemic (due to 'common cause') defects are reduced quickly and in volume (2/3 in year)**
- **The inside 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 work, 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.**

Summary

- **2/3 annual defect reduction**
- **Exploits practical shop floor knowledge**
- **Suggested changes are acceptable**
- **Empowered Creativity**
- **Many small practical improvements**
- **DPP can be proven locally as a pilot**

DPP Policy



Detail

- **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.**

Summary

- **Frequent local analysis**
- **Eat your own dog food (try ideas)**
- **Spread good stuff fast (ex. Douglas Templates)**
- **5% investment**
- **Invest NOW, early in project**

Cost of Quality over Time: Raytheon 95

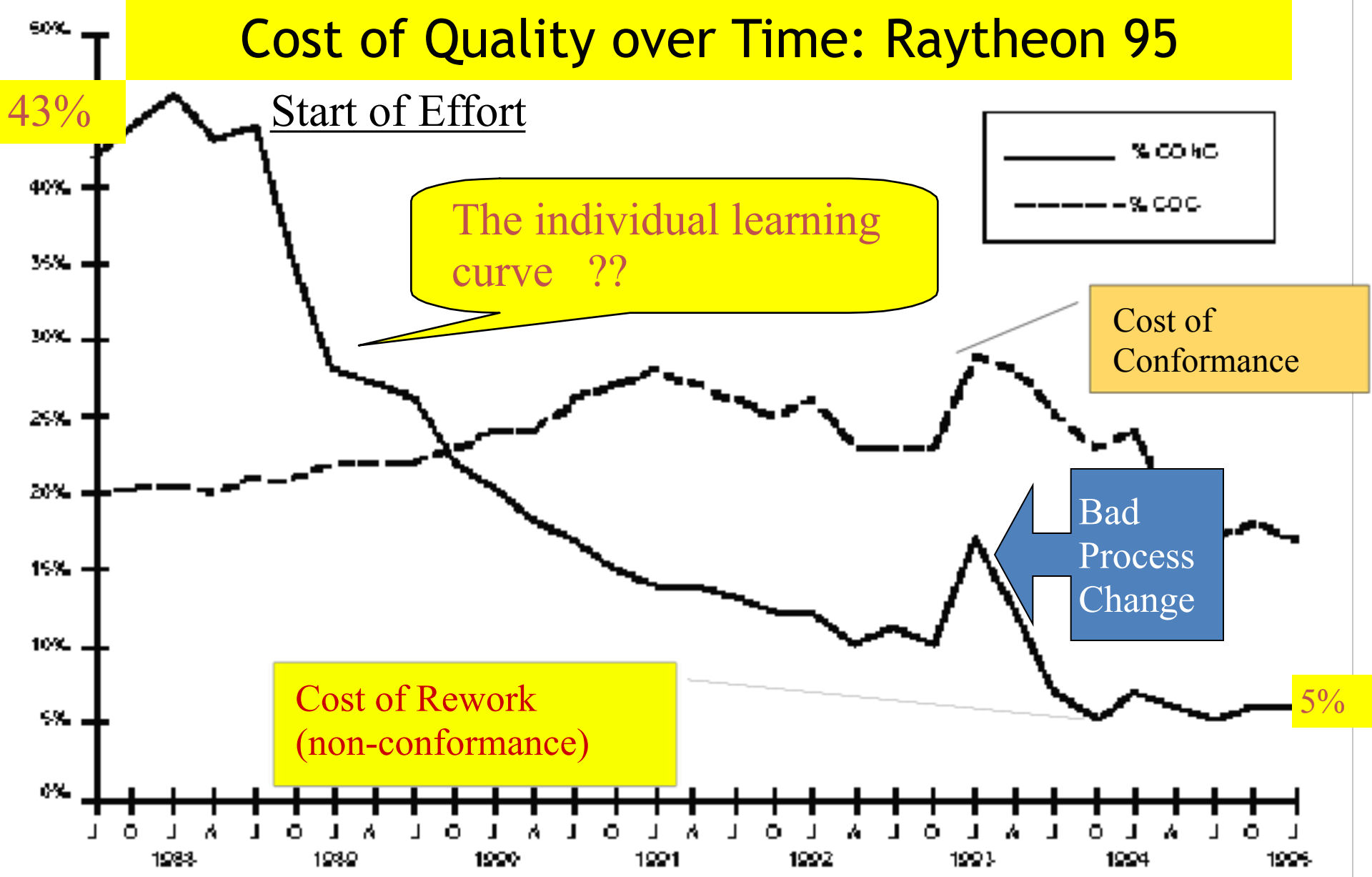
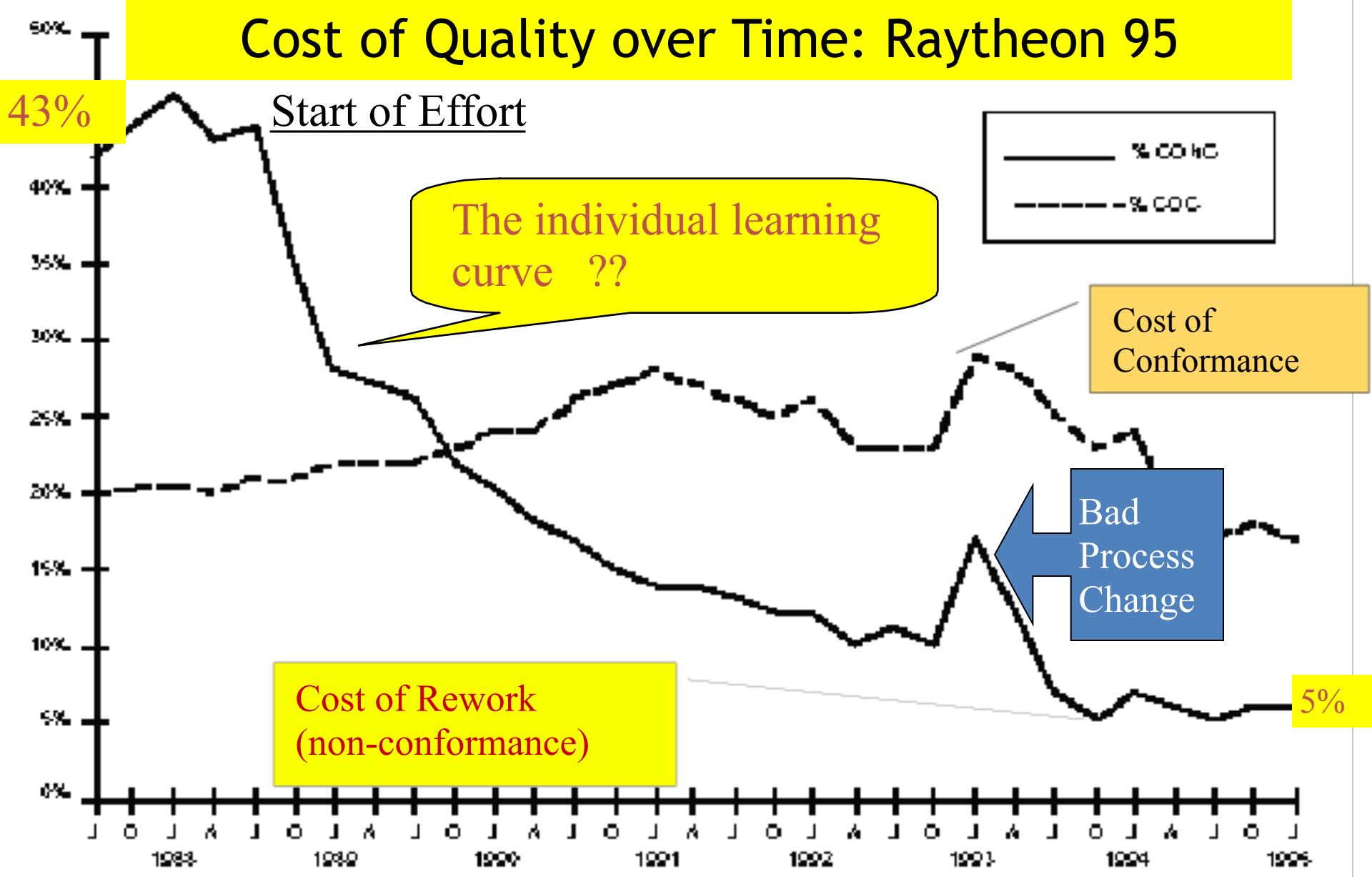


Figure 8: Cost of Quality Versus Time

Cost of Quality over Time: Raytheon 95



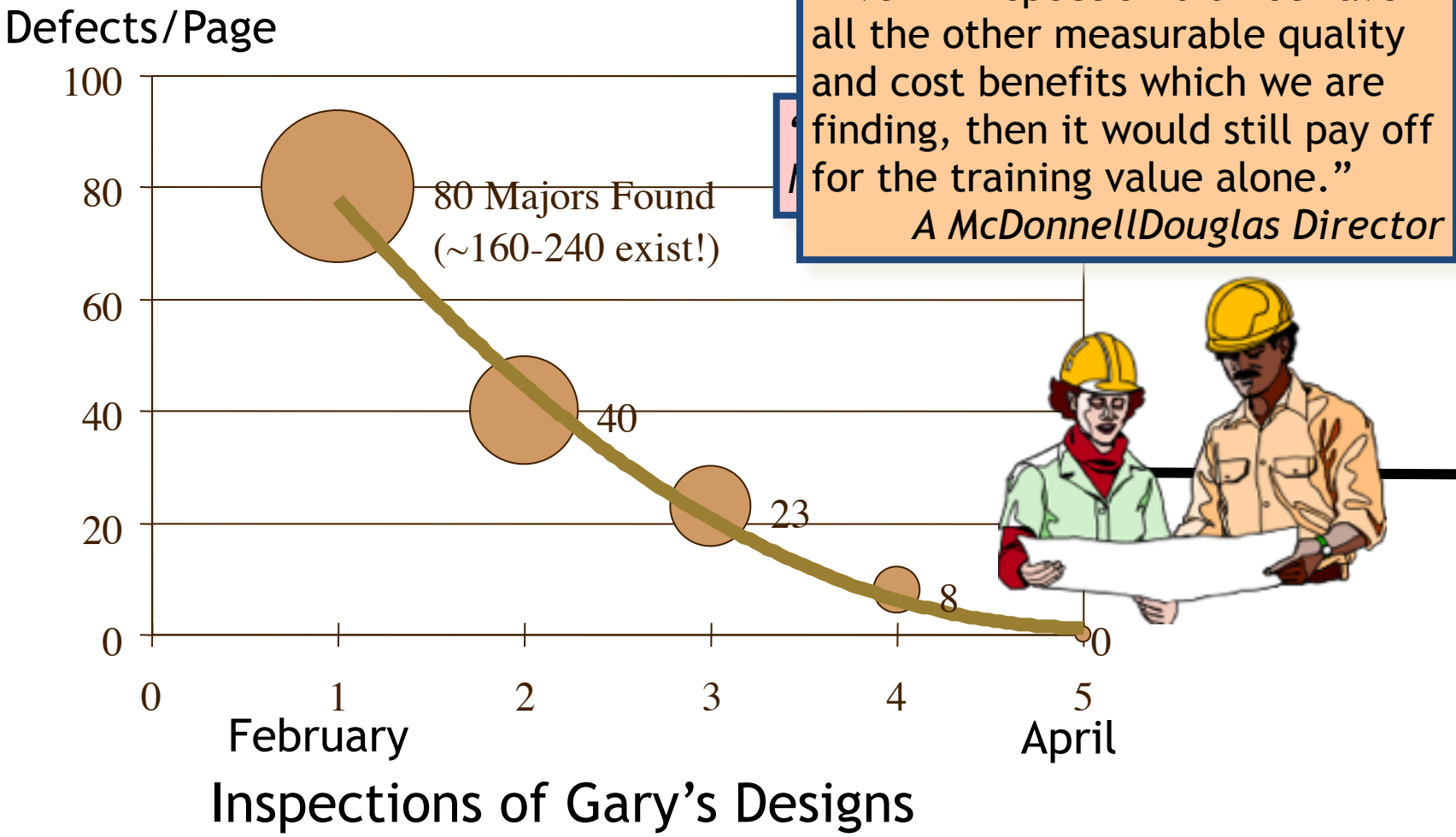
www.sei.cmu.edu/reports/95tr017.pdf

Figure 8: Cost of Quality Versus Time

Positive Motivation
Personal Improvement

"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



Inspections of Gary's Designs

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

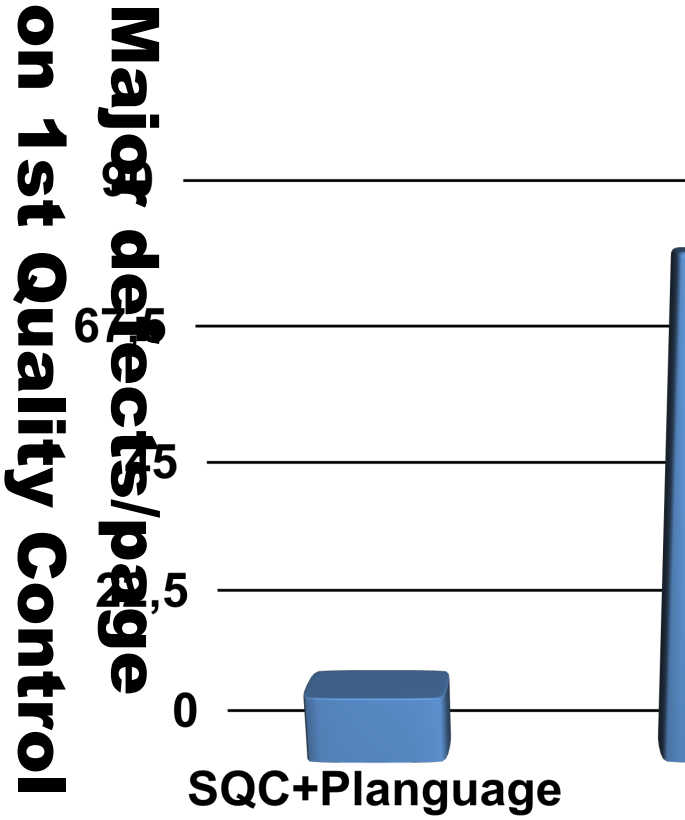
Across 18 DV (DeVelopment) Projects using the new requirements method, the average major defect rate on first inspection is 11.2.

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:

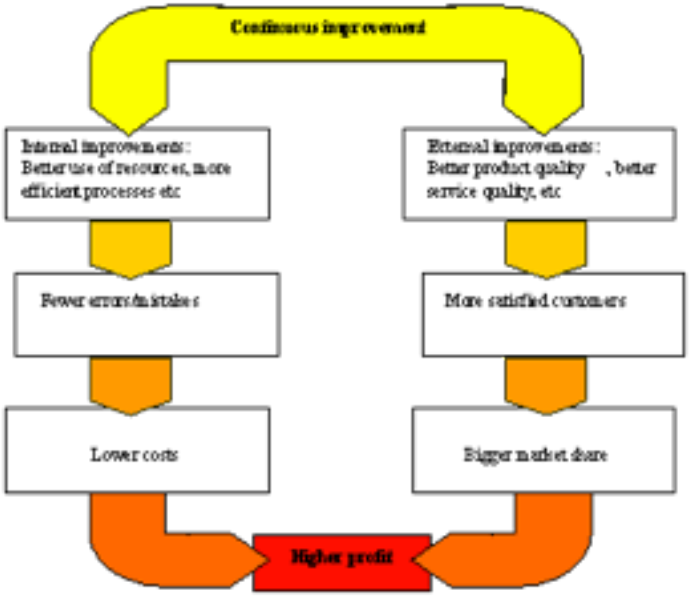
- **The requirement is uniquely identifiable**
- **All stakeholders are identified.**
- **The content of the requirement is 'clear and unambiguous'**
- **A practical test can be applied to validate it's delivery.**

The average major defect rate in this sample was 80.4.

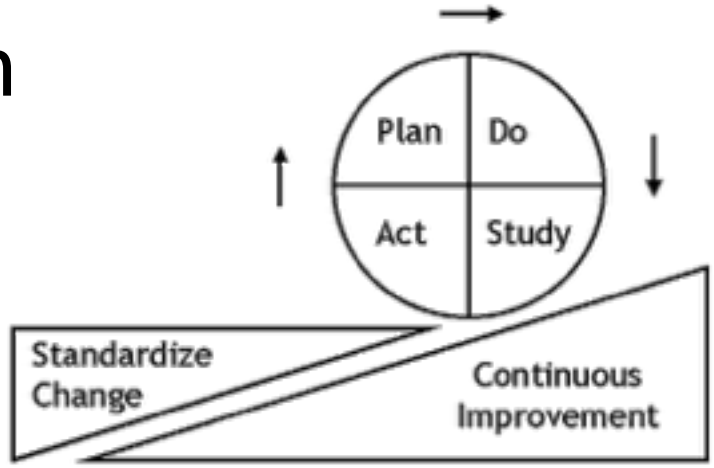


Defect Detection strategies versus Defect Prevention strategies

- 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



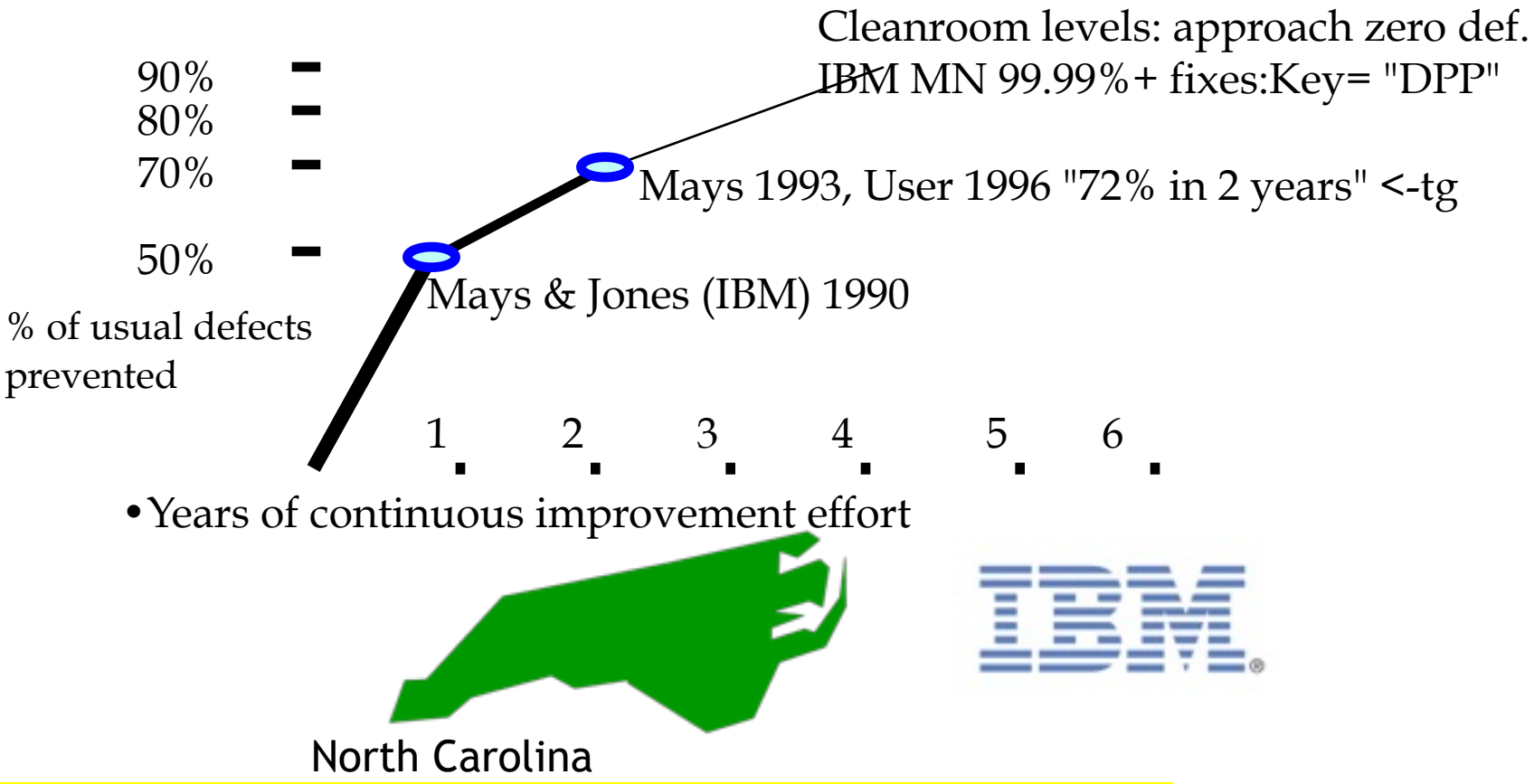
Prevention Costs



Deming Cycle

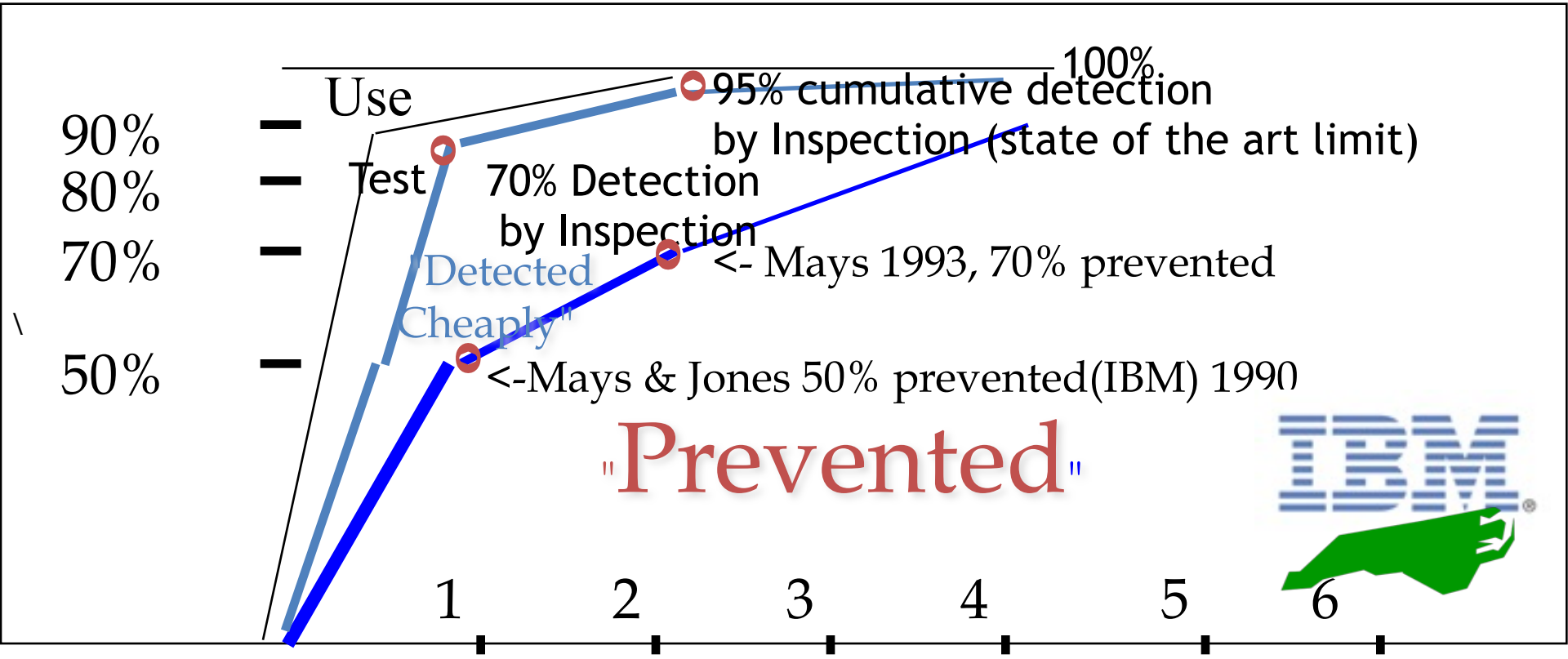
- 5%, stable at 5%
  - of development costs
  - (Raytheon 1993)
- 0.5 % of development costs
  - (Mays 1995)

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



IBM Research Triangle Park Networking Laboratory

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)

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

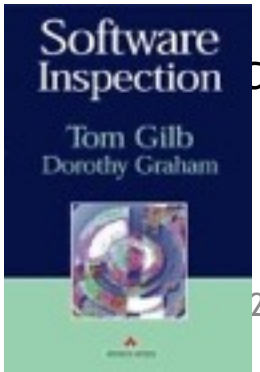
Defect Prevention Process (DPP)

Papers

- **Raytheon** Electronic Systems Experience in Software Process Improvement
  - Technical Report CMU/SEI-95-TR-017, ESC, TR-95-017
  - www.sei.cmu.edu/reports/95tr017.pdf

- Experiences with Defect Prevention. (1990) at **IBM** Systems Journal , Robert G, Mays
  - http://agileconsortium.pbworks.com/f/Mays1990ExperiencesDefectPreventionIBMSysJ.pdf
    - R Mays IBM SJ Paper on Defect Prevention Process, DPP

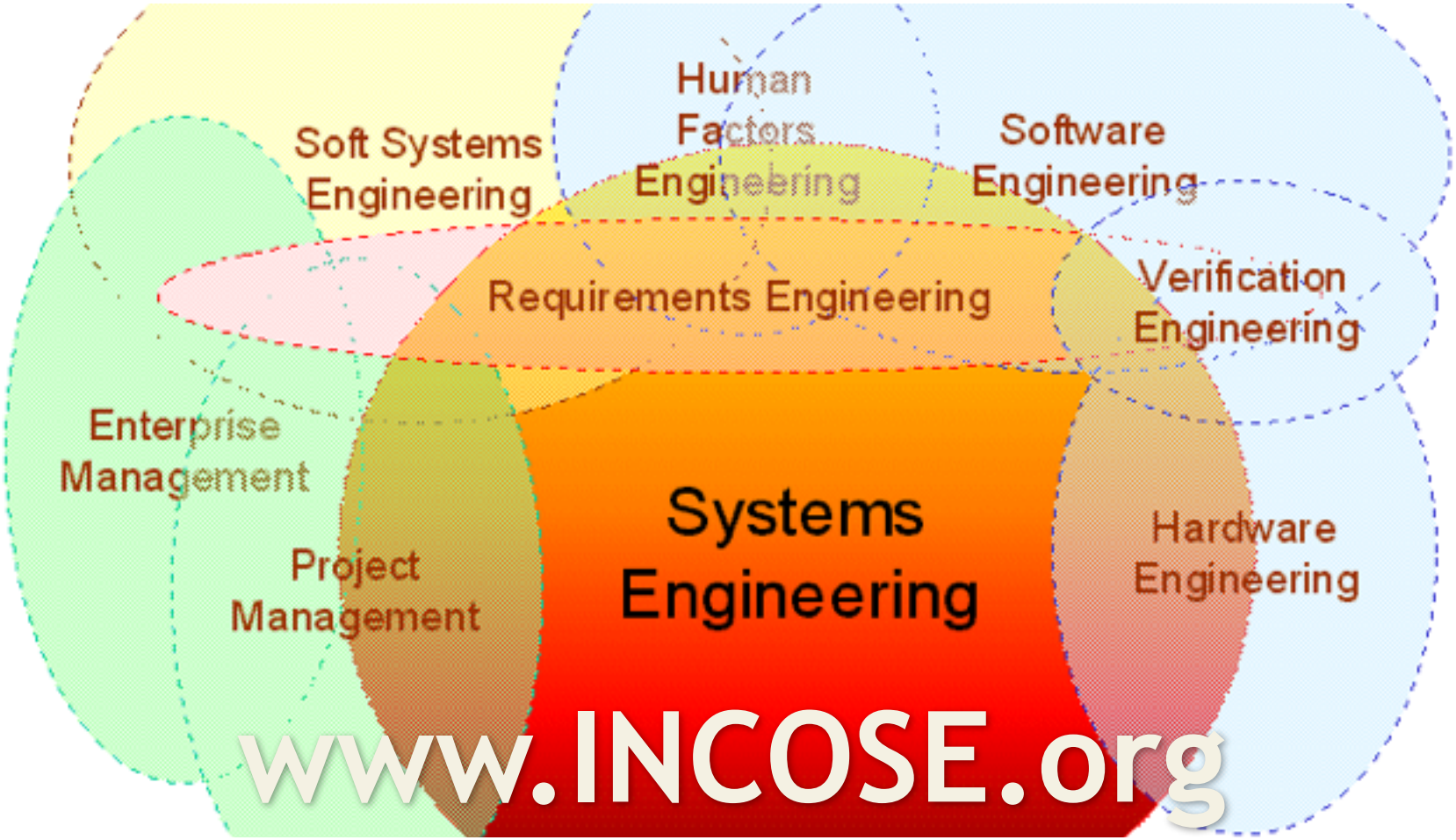
Chapter 17 in Gilb SW Inspection



Case Aircraft Mfgr.

- DAC Case:
  - The experiences of implementations of Gilb's Inspection methods on large scale quality control of engineering drawings and engineering orders.
  - http://www.gilb.com/tiki-download_file.php?fileId=254
  - http://www.gilb.com/tiki-download_file.php?fileId=253

Systems Engineering (for software projects)



Systems level

Not

- Working code to the customer
- Software Engineer (=coder)

But this

- Measurable value delivery to the stakeholder
- Everything managed that is necessary to actually deliver value
  - Motivation, information, training, data, contractds, hardware, and maybe…. code

‘No Cure, No Pay’ Contracting, and Project Management

- This is NOT a widespread practice. But it should be, and I have done it to a limited extend for decades in practice.

My suggestion is simple.

- **Pay only when defined results are provably delivered.**

- **This requires several things:**
  - **o Contracts that release payment only for meaningful results.**
  - **o The ability to define those results,**
    - **particularly qualitative ones,**
    - **and particularly organizational ones.**

- **The ability to deliver those results incrementally,**
  - **thus proving capability at early stages and continuously.**
  - **And being able to cancel bad suppliers early**
  - **Allowing work in parallel with multiple suppliers**

Defining the 'Cure'

We write contracts, and we write requirements for projects,
but these are <u>normally useless</u> for the following reasons.

- **We define the wrong things**

- **We define (valueless things! Like)**
  - **Designs, architectures, technologies (not results of them)**
  - **Functions and use cases: not the improvements and benefits to them**
  - **Hours of effort, not value delivered**
  - **The 'names' of critical benefits ('higher productivity')**

but we do not define them measurably.

- **We fail to define:**
  - **'value'**
    - **The dozens of stakeholders involved**
    - **The results that the stakeholders value**
    - **The quality levels, numerically and measurably**
    - **The knock-on effects of the new or improved system,**

expected at a higher level

- **A series of early, short, and frequent value delivery stages**

We can evaluate potential suppliers ‘dynamically’..

- **We can instead choose suppliers based on**
  - **proven ability in the past,**
  - **and on our project, to deliver.**

- **We can, as I suggested to a UK Government Agency,**
  - **allow 3 competing contractors to start work in parallel,**
  - **and move work towards the ones that prove their ability to deliver value.**
  - **Move work away from those that do not deliver value as promised.**
  - **If all 3 perform well, fine, keep them going!**
  - **They would be working on complimentary aspects of the system.**

The Request For Proposal. RFP.
(Example of components)

The request for proposal will sound like this:

1. **"We invite you to tender for a contract to <build software/deliver an IT system>.**
2. **The contract will be based on a Value Payment system.**
3. **This means that we will define what we expect in terms of testable and measurable values from the system.**
4. **We will pay only when that value is satisfactorily and provably delivered.**
5. **We will not pay for effort put in,**
6. **and we will not pay for sub-specification results.**
7. **If you are focused on delivering us the results we agree on, then you can earn money independently of the costs to you.**
8. **Efficient suppliers can earn more than usual.**
9. **Inefficient suppliers would not.**
10. **We hope you will get rich by helping us to get what we expect for our money."**

Specifying the Contract.

- **The contract can be as simple as the No Cure No Pay contract template in the next slide.**
  - **It is a framework for sub-contracting at the Evolutionary Value Delivery step level.**

- **<u>The essential ideas in a No Cure No pay Contract are:</u>**

- **1 Payment is totally dependent on proven delivery of our Value Definition.**

- **2 Estimates, for delivering the value, will be made by the Contractor, in advance**

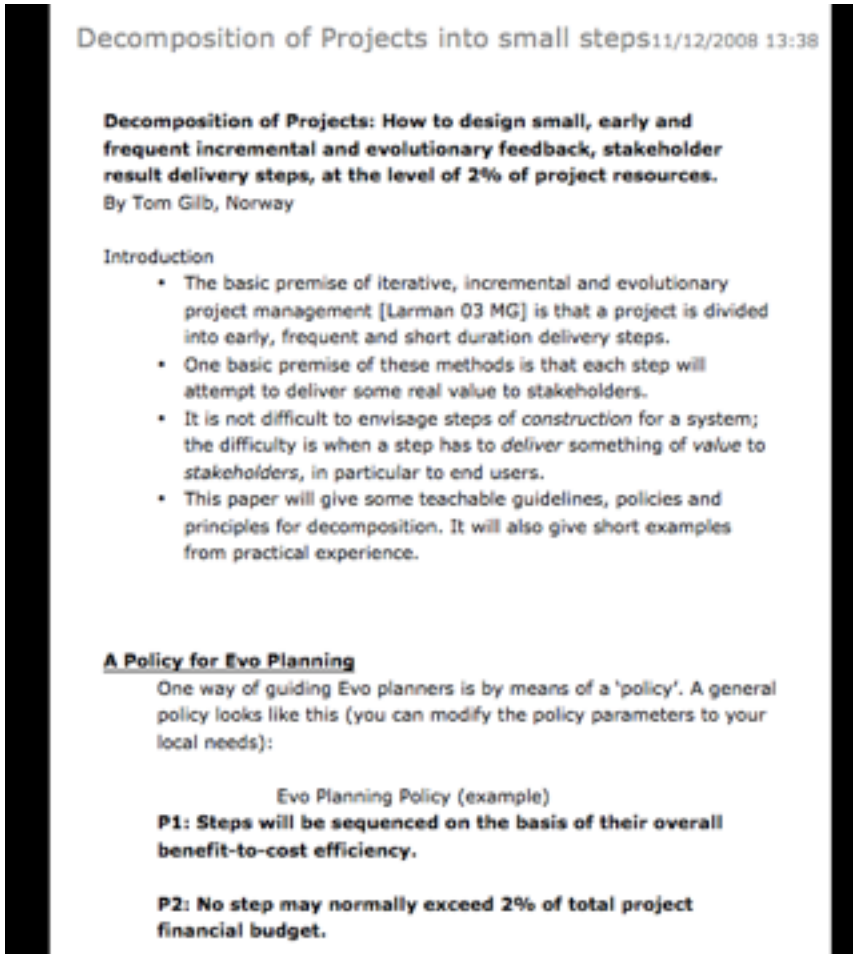
- **3 We will accept some level of cost overrun,**
  - **compared to the estimates,**
  - **when actual costs exceed the estimate.**
    - **Example 100%. Above that, the Contractor pays such excess costs.**

- **4 We will allow invoicing**
  - **to be triggered based on a simple test of delivery.**
  - **Actual payment of the invoice is dependent on**
    - **a trial period with continued success.**
      - **For example 30 days.**

Sample Contract Addition; developed for The Law Society by TG

Drop this into a Conventional Contract
• Author Tom Gilb .
•
• **Contract Design idea**: designed to work within the scope of a present contract with minimum modification.
An Evo step is considered a step on the path to delivering a phase.
You can choose to declare this paragraph has priority over conflicting statements (30.1),
or to clean up other conflicting statements in the initial contract basis.
•
§30. Evolutionary Result Delivery Management.
• 30.1 **Precedence**. This paragraph has precedence over conflicting paragraphs.
• 30.2 **Steps of a Phase.** The Customer may optionally undertake to specify, accept and pay for evolutionary usable increments of delivery, of the defined Phase, of any size. These are hereafter called "**Steps**".
• 30.3 **Step Size.** Step size can vary as needed and desired by the Customer, but is assumed to usually be based on a regular weekly cycle duration.
• 30.4 **Intent**. The intent of this evolutionary project management method is that the Customer shall gain several benefits: earlier delivery of prioritized system components, limited risk, ability to improve specification after gaining experience, incremental learning of use of the new system, better visibility of project progress, and many other benefits. This method is the best known way to control software projects [Larman03].
• 30.5 **Specification Improvement.** All specification of requirements and design for a phase will be considered a framework for planning, not a frozen definition. The Customer shall be free to improve upon such specification in any way that suits their interests, at any time. This includes any extension, change or retraction of framework specification which the Customer needs.
• 30.6 **Payment for Acceptable Results**. Estimates given in proposals are based on initial requirements, and are for budgeting and planning purposes. Actual payment will be based on successful acceptable delivery to the Customer in Evolutionary Step deliveries, fully under Customer Control. The Customer is not obliged to pay for results which do not conform to the Customer-agreed Step Requirements Specification.
• 30.7 **Payment Mechanism**. Invoicing will be on a Step basis triggered by end of Step preliminary (same day) signed acceptance that the Step is apparently as defined in Step Requirements. If Customer experience during the 30 day payment due period demonstrates that there is a breach of specified Step requirements, and this is not satisfactorily resolved by the Company, then a Stop Payment signal for that Step can be sent and will be respected until the problem is resolved to meet specified Step Requirements.
• 30.8 **Invoicing Basis**. The documented time and materials will be the basis for invoicing a Step. An estimate of the Step costs will be made by
• the Company in advance and form a part of the Step Plan, approved by the Customer.
• 30.9 **Deviation**. Deviation plus or minus of up to 100% from Step cost and times estimates will normally be acceptable (because they are small in absolute terms), as long as the Step Requirements are met. (The Customer prioritises quality above cost). Larger deviations must be approved by the Customer in writing before proceeding with the Step or its invoicing.
• 30.9 **Scope**. This project management and payment method can include any aspect of work which the Company delivers including software, documentation and training, maintenance, testing and any requested form of assistance.

Decomposition Principles
A Teachable Discipline



Decomposition of Projects into small steps11/12/2008 13:38

Decomposition of Projects: How to design small, early and frequent incremental and evolutionary feedback, stakeholder result delivery steps, at the level of 2% of project resources.

By Tom Gilb, Norway

Introduction

- The basic premise of iterative, incremental and evolutionary project management [Larman 03 MG] is that a project is divided into early, frequent and short duration delivery steps.
- One basic premise of these methods is that each step will attempt to deliver some real value to stakeholders.
- It is not difficult to envisage steps of *construction* for a system; the difficulty is when a step has to *deliver* something of *value* to *stakeholders*, in particular to end users.
- This paper will give some teachable guidelines, policies and principles for decomposition. It will also give short examples from practical experience.

A Policy for Evo Planning

One way of guiding Evo planners is by means of a 'policy'. A general policy looks like this (you can modify the policy parameters to your local needs):

Evo Planning Policy (example)

P1: Steps will be sequenced on the basis of their overall benefit-to-cost efficiency.

P2: No step may normally exceed 2% of total project financial budget.

How to decompose systems into small evolutionary steps::

1• *Believe* there is a way to do it, you just have not *found* it yet!

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 user or customer, however small.

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 customer, tomorrow, one interesting improvement.

6• Focus on the *results* (which you should have defined in your goals, moving toward target 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 customer won't like it. *If* you are f
they want, then by definition, *they* should like it. If you are n

10• Don't get so worried about "what might happen afterward
make no practical progress.

11• You cannot foresee everything. Don't even *think* about it!

12• If you focus on helping your customer in practice, *now*, w
need it, you will be forgiven a lot of 'sins'!

13• You can understand things much better, by getting *some*
experience (and removing *some* of your fears).

14• Do *early* cycles, on willing local mature parts of your user 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.

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' or end user. They probably have dozens of suggestions.

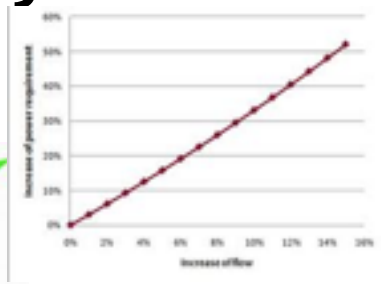
19• Talk with end users in *any* case, they have insights you need.

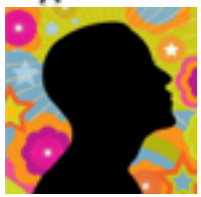
20• Don't be afraid to use the old system and the old 'culture' as a launching platform for the radical new system. There is a lot of merit in this, and many people overlook it.

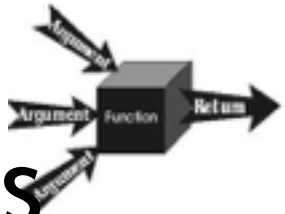
http://www.gilb.com/tiki-download_file.php?fileId=41

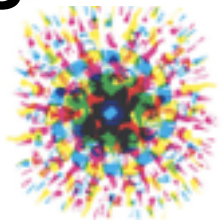
1 1 1 1 1 1 Unity

- –1% increase at least
- –1 stakeholder
- –1 quality or value
- –1-week delivery cycle
- –1 function focus
- –1 design used









12 Tough questions:© Tom Gilb

- **1. Numbers**
  Why isn't the improvement quantified?

  2. Risk
  What is the degree of risk or uncertainty, and why?

  3. Doubt
  Are you sure? If not, why not?

  4. Source
  Where did you get that information? How can I check it out?

  5. Impact
  How does your idea effect my goals and budgets, measurably?

  6. All critical factors
  Did we forget anything critical to survival?

- **7. Evidence**
  How do you know it works that way? Did it 'ever'?

  8. Enough
  Have we got a complete solution? Are all requirements satisfied?

  9. Profitability first
  Are we planning to do the 'profitable things' first?

  10. Commitment
  Who is responsible for failure, or success?

  11. Proof
  How can we be sure the plan is working, during the project, early?

  12. No cure, no pay
  Is it 'no cure, no pay' in a contract? Why not?

Rene Descartes on Focus

- **"We should bring the whole force of our minds**
  - **to bear upon the most minute and simple details**
  - **and to dwell upon them for a long time**
  - **so that we become accustomed to perceive the truth clearly and distinctly."**
- **Rene Descartes, Rules for the Direction of the Mind, 1628**

Tao Te Ching (500BC)

- That which remains quiet, is easy to handle.
- That which is not yet developed is easy to manage.
- That which is weak is easy to control.
- That which is still small is easy to direct.
- Deal with little troubles before they become big.
- Attend to little problems before they get out of hand.
  - For the largest tree was once a sprout,
- the tallest tower started with the first brick,
- and the longest journey started with the first step.

From Lao Tzu in Bahn, 1980 (also quoted in Gilb, Principles of Software Engineering Management page 96), Penguin book

‘No Cure, No Pay’ Contracting, and Project Management: References

- Slides: No Cure
- http://www.gilb.com/tiki-download_file.php?fileId=85

- Paper No Cure
- http://www.gilb.com/tiki-download_file.php?fileId=38

- 12 Tough Questions
- **A full paper is available**
- **http://www.gilb.com/tiki-download_file.php?fileId=24**

- Decomposition by Value

- Decomposition of Projects: How to Design Small Incremental Steps INCOSE 2008 Paper
  - http://www.gilb.com/tiki-download_file.php?fileId=41

- Decomposition Slides Aug 2010
  - http://www.gilb.com/tiki-download_file.php?fileId=350

- 111111 Unity Method slides 10 minute Talk
  - http://www.gilb.com/tiki-download_file.php?fileId=451

Specification Quality Control: SQC

Case:
Real Agile Spec QC

- of System Requirements Specification

(SRS) of 82 pages for a major US

corporation.

Framework

- Demonstration of power of Agile Inspection
  - 8 Managers
  - 2 hours
  - 4 real requirements specifications offered ,
    - One 82 page ‘System Requirements Specification’ actually used

We Introduced best-practice Rules for Requirements

- 1. Unambiguous to intended Readership
- 2. Clear enough to test.
- 3. No unintentional Design

We explained the definition of Spec Defect (1)

- A 'Specification Defect' is a *violation of a Specification Rule*
  - *—(violation of a 'standard')*
  - *Note: If there are 10 ambiguous terms in a single requirement*
    - *then there are 10 defects!*

We further explained the definition of Spec Defect (2)

- **A 'Specification Defect' is a also a 'Potential Defect'**
  - **In the next level of specification**
    - **Like in Design, test plans, code or test scripts.**
  - **With about 1/3 chance (potential) of becoming a downstream real defect.**
  - **In 'code' we call this a 'bug' (a potential malfunction).**
  - **If we discover the bug in test or operation we call it a 'malfunction'**

The definition of Major defect

- Major:
  - a Defect that *potentially*

costs more

- to find and fix
- later in the development process
- than it would cost *now*.
- We need to get rid of it NOW!

The downstream alternative cost of quality at a UK Defence Electronics Factory.

9 to 1 more

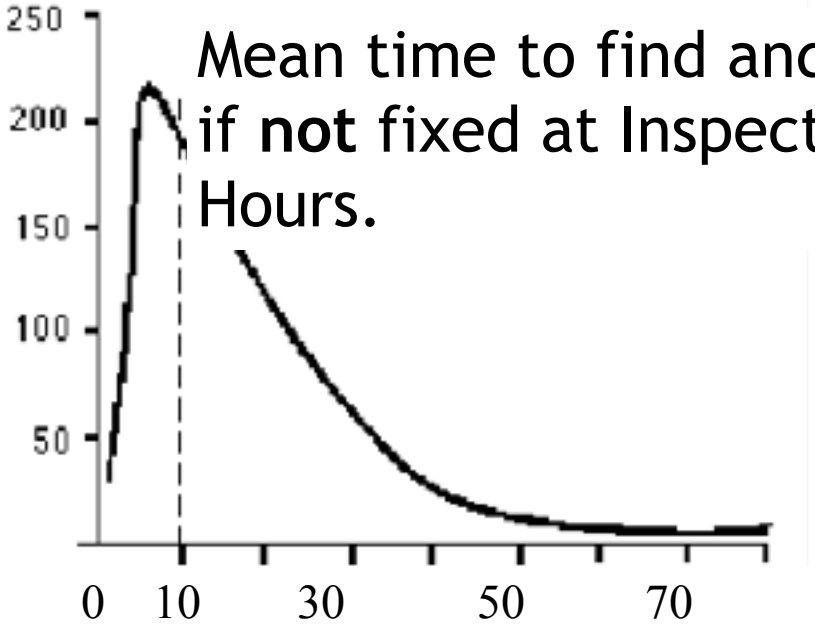
(all types of documents for electronics).

Number of defects of the 1,000 sampled Majors ------>

That we

manually estimated

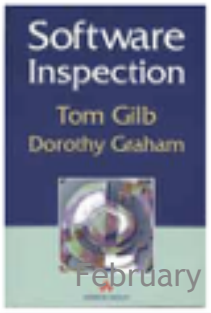
downstream costs to fix



Mean time to find and correct a Major if **not** fixed at Inspection was **9.3** Hours.

It cost about **1** hour to find and fix a Major **at** time of Inspection

Trevor Reeve



Source: Trevor Reeve, Case Study Chapter in "Software Inspection", Gilb client.

Philips MEL became "Thorn EMI", then Racal. Crawley UK. 1999 Raytheon

Agree with Management on Exit level

- Exit Conditions: (when Requirements can go forward to Design, Test, etc. with little risk)
  - Maximum 1 Major Defect/ (Logical) Page, estimated remaining
  - Logical Page = 300 Non-commentary words.

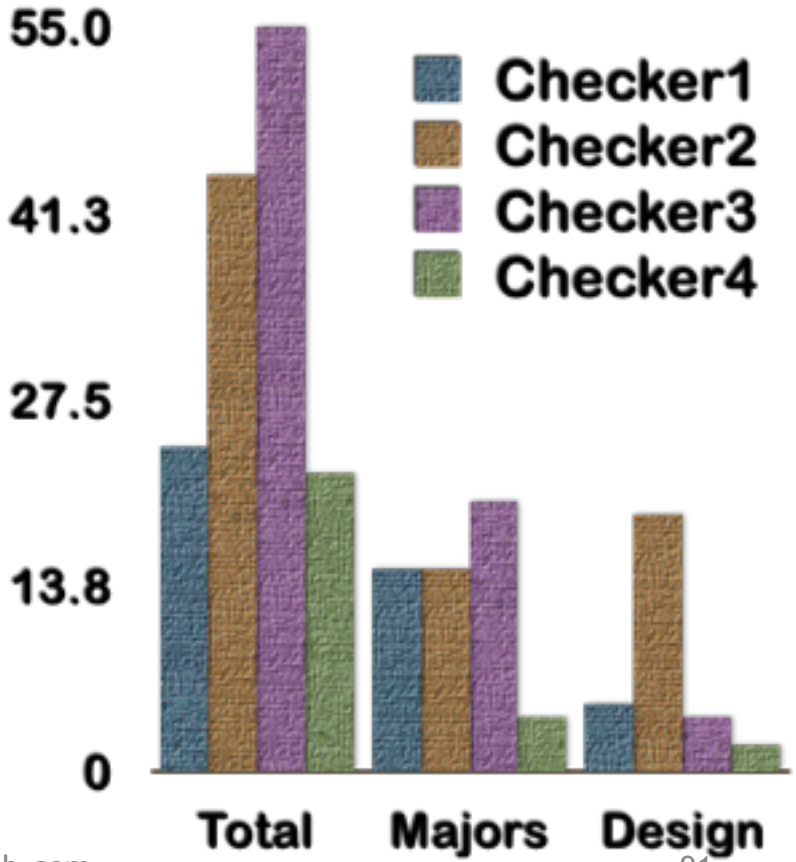
The notion of 'numeric exit': When are requirements 'good enough' to build tests on ?

- A major defect in requirements has 33% chance of causing a bug or worse.
- Most IT shops are 'uncontrolled': have no standards enforced, no QC of requirements
- They have about 100 ±50 major defects per page
  - 33 ± 10 potential bugs per page
- This is deemed completely unacceptable by managements, and in relation to cost and quality options
- This (max. 1 major/page) should be your requirements process exit standard, and also your test planning entry standard

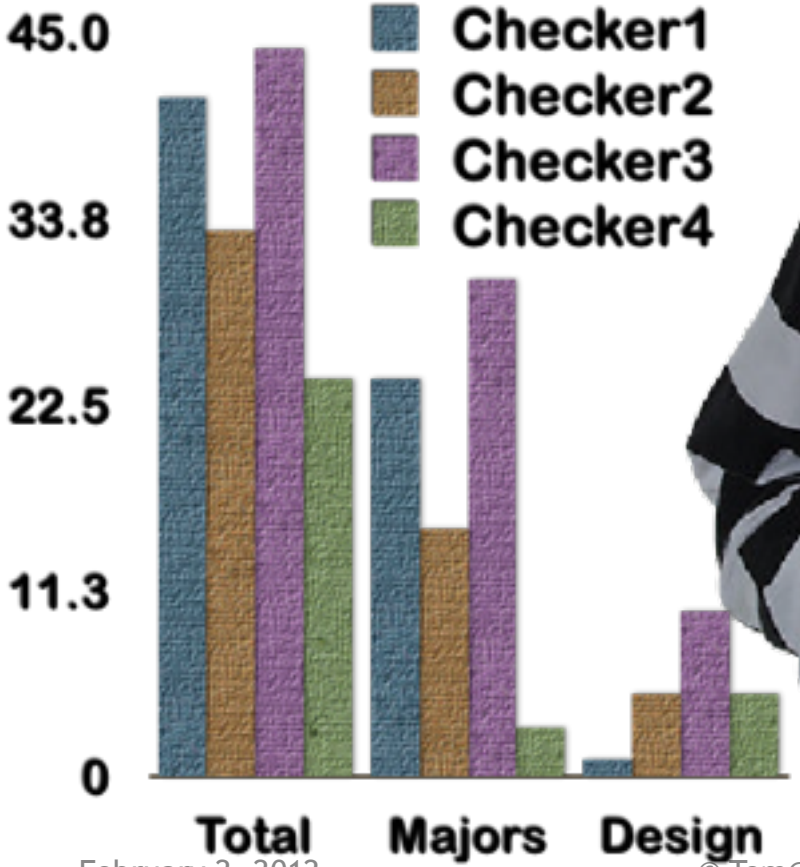
The assigned checking process

- You have up to 30 minutes
  - check 1 *sample* requirements page (from an 82 page document)
- Count all *potential* Rule Violations
  - = Defects
- *Classify* Defects as Major or minor

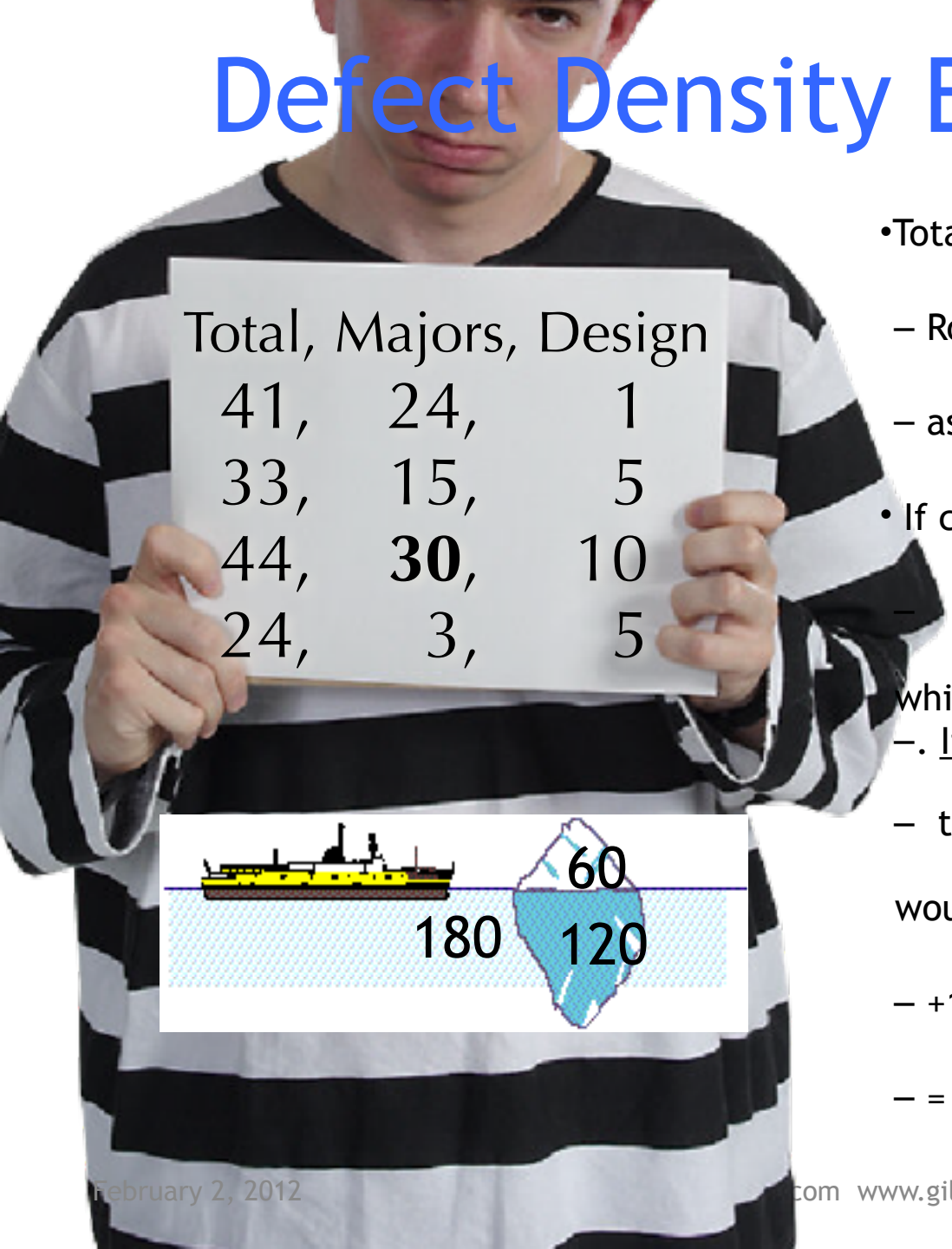
Report Page 81



Report
Page 82



Defect Density Estimation



- Total for group (page 82)
  - Rough Est. 30 x 2 = 60 Majors
  - assume 60 ±10 are unique.
- If checking is 33.33% effective,
  - total in page = 3 x 60 = about 180±30 Of which 2/3 (or 120) were not yet found.
  - . If we fix all we found (60),
  - then the estimated remainder of Majors would be 120 (not found)
  - +10 "not fixed correctly"
  - = 130 Majors remaining.

Conclusions

- Human *defect removal* by Inspections/reviews/SQC is
  - a hopeless cause: not worth it.
- Spec QC *can* be used, in spite of imperfect effectiveness,
  - to '*accurately enough' estimate* 'major-defect-level' density.
- EXPERIENCE AND CONSEQUENCES:
- This measurement can be used to motivate engineers to
  - dramatically (100x! Over about 7 learning cycles)
    - reduce their defect insertion (rule violation)
  - to a *practical* exit level
    - » (like *less than* 1.0 Majors/page)

Extrapolation to Whole Document

- Average: 150 Majors/page
  - Page 81: 120 majors/page
  - Page 82: 180 Majors/page
- Total in whole document:
  - 12,300 Majors
  - 150 Majors/page x 82 pages.

Estimated Project Loss

- If a Major has
  - 1/3 chance of causing loss
- And each *loss* caused by a Major is
  - avg. 10 hours
  - then total project Rework cost is
  - about *41,000 hours loss.*
- (This project <u>was</u> over a year late)
  - 1 year = 2,000 hours x 10 people

P1: Identify Checkers: Two people, maybe more, should be identified to carry out the checking.

P2: Select Rules: The group identifies about three rules to use for checking the specification. (My favorites are clarity ('clear enough to test'), unambiguous ('to the intended readership') and completeness ('compared to sources'). For requirements, I also use 'no optional design'.)

P3: Choose Sample(s): The group then selects sample(s) of about one 'logical' page in length (300 non-commentary words). Choosing such a page at random can add *credibility* – so long as it is *representative* of the content that is subject to quality control. The group should decide whether all the checkers should use the same sample, or whether *different* samples are more appropriate.

P4: Instruct Checkers: The SQC team leader briefly instructs the checkers about the rules, the checking time, and how to document any defects, and then determine if they are *major* defects (majors).

P5: Check Sample: The checkers use between 10 and 30 minutes to check their sample against the selected rules. Each checker should 'mark up' their copy of the document as they check (underlining issues, and classifying them as 'major' or not). At the end of checking, each checker should count the number of 'possible majors' (spec defects, rule violations) they have found in their page.

P6: Report Results: The checkers each report to the group their number of 'possible majors.' Each checker determines their number of majors, and reports it.

P7: Analyze Results: The SQC team leader extrapolates from the findings the number of majors in a single page (about 6 times** the most majors found by a single person, or alternatively 3 times the unique majors found by a 2 to 4 person team). This gives the major-defect density estimate. If using more than one sample, you should average the densities found by the group in different pages. The SQC team leader then multiplies the 'average major defects per page density' by the 'total number of pages' to get the 'total number of major defects in the specification' (for dramatic effect!).

P8: Decide Action: If the number of majors per page found is a large one (ten majors or more), then there is little point in the group doing anything, except determining how they are going to get someone to write the specification 'properly', meaning to acceptable exit level. There is no economic point in looking at the other pages to find 'all the defects', or correcting the majors already found. There are simply too many majors not found.

P9: Suggest Cause: The team then chooses any major defect and thinks for a minute why it happened. Then the team agrees a short sentence, or better still a few words, to capture their verdict.

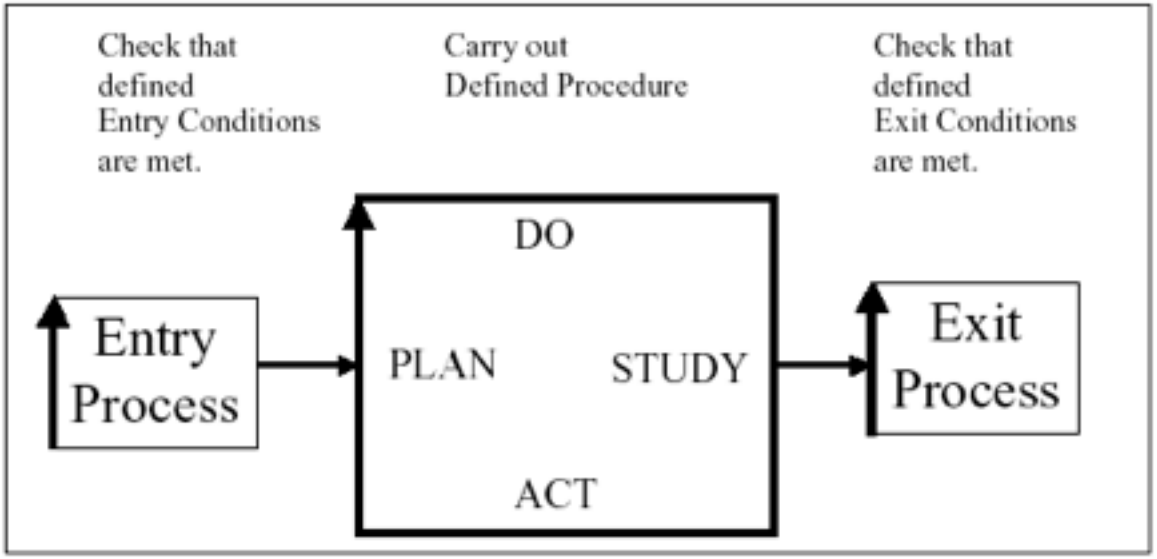
The formal Agile SQC Process Sources

- 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

Quantified Software *Process* Control: Numeric XE

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



- 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!

Life Cycle Engineering of System and Product *Adaptability*

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 H-project requirement in all its uses, it is especially critical in our applications where the much longer job durations afford software defects (e.g. memory leaks) a greatly expanded opportunity to surface.**
- **In this regard,**
- **H-project will provide the following features or attributes:**
  - **Minimal down-time**
- **A critical H-project objective is to have minimal downtime due to software failures.**
- **This objective includes:**
  - **Mean time between forced restarts > 14 days**
- **H-project'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**
- **H-project 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.*

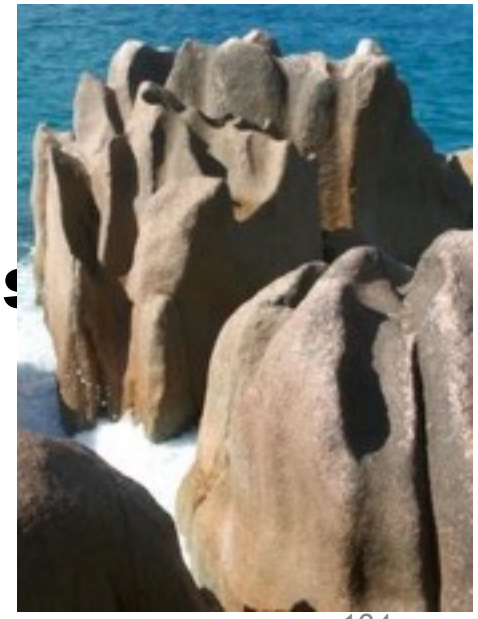
Real case of requirement for project costing over $100,000,000 without delivering testable results

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 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

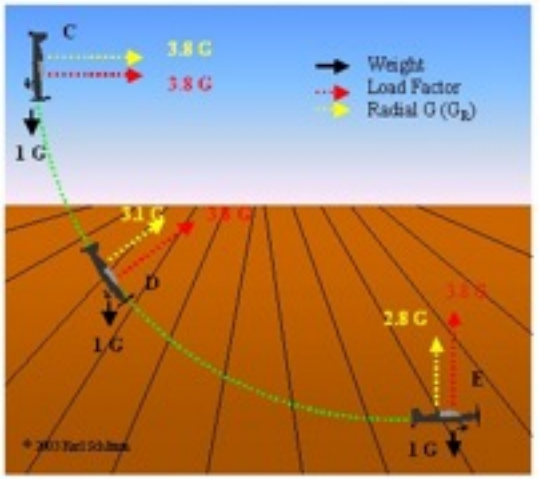
Goal [By 2008?, Activity = Data Acquisition, Intensity = Lowest level] : 300 days ??

Stretch: 600 days

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.

Testability:

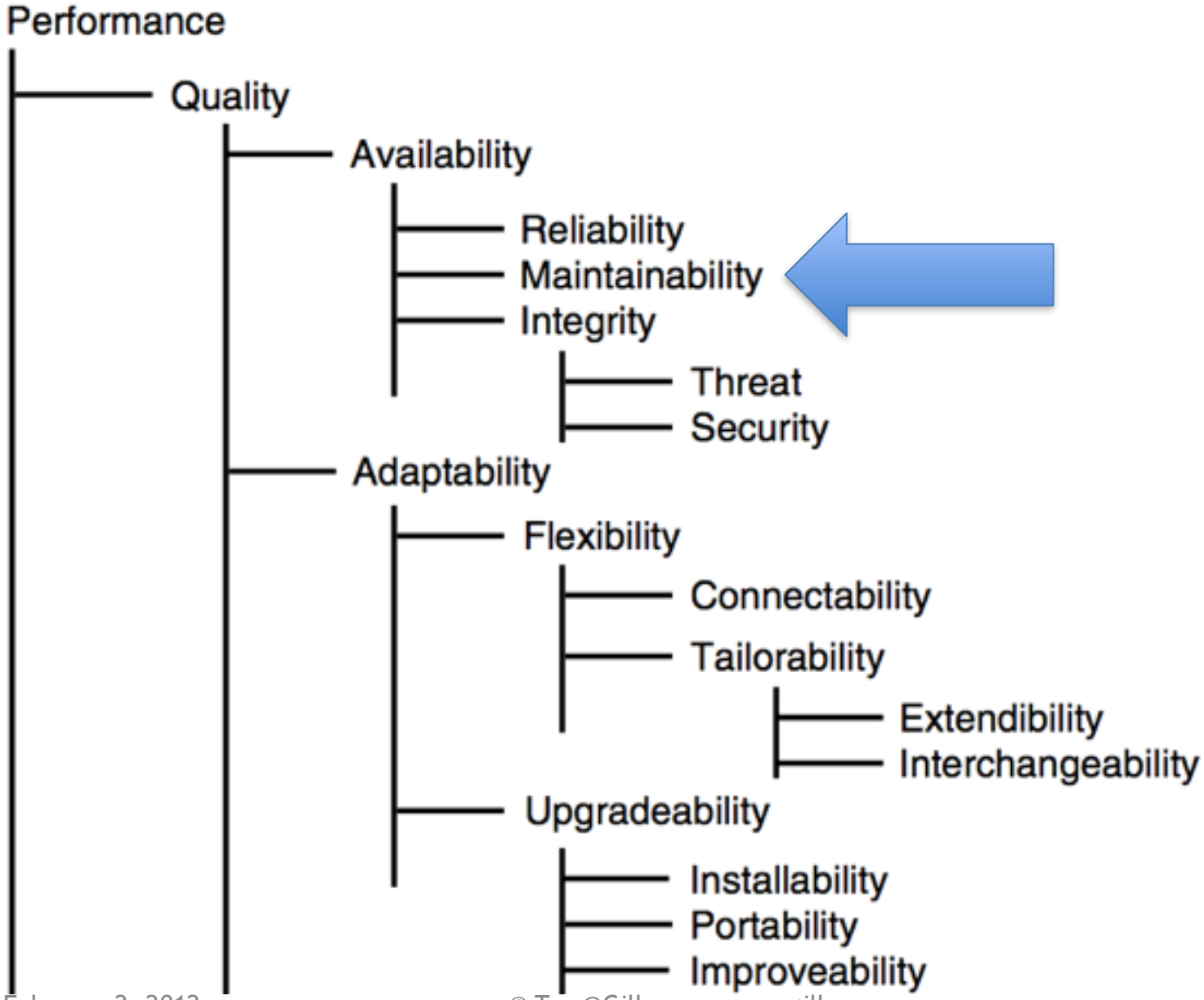
Testability:
Type: Software Quality Requirement.
Version: 20 Oct 2006-10-20
Status: Demo draft,
Stakeholder: {Operator, Tester}.
Ambition: Rapid-duration automatic testing of <critical complex tests>, with extreme operator setup and initiation.

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*

Broader Maintainability Concepts



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?

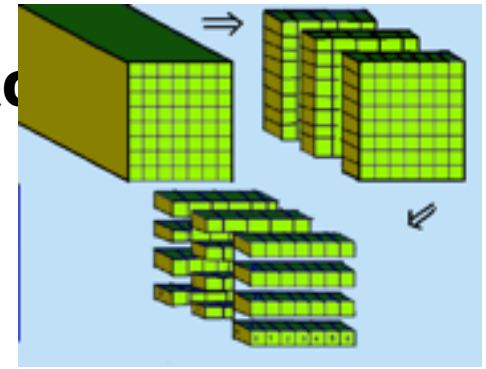
3. **Tool Collection Time.**

   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 Time**
11. **Customer Installation Time**
12. **Customer Damage Analysis Tim**
13. **Customer Level Recovery Time**
14. **Customer QC Time**



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* (Ireson 1966).

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

With Scale Templates

DoDef. Persinscom Impact Estimation Table: MULTIPLE -ilities

Designs

Requirements

R→ D Impacts

Design Ideas ->	*Technology Investment*	*Business Practices*	*People*	*Empowerment*	*Principles of IMA Management*	*Business Process Re-engineering*	*Sum Requirements*
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*	

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'**
- **most of the types maintenance (f**

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.

Flexibility:

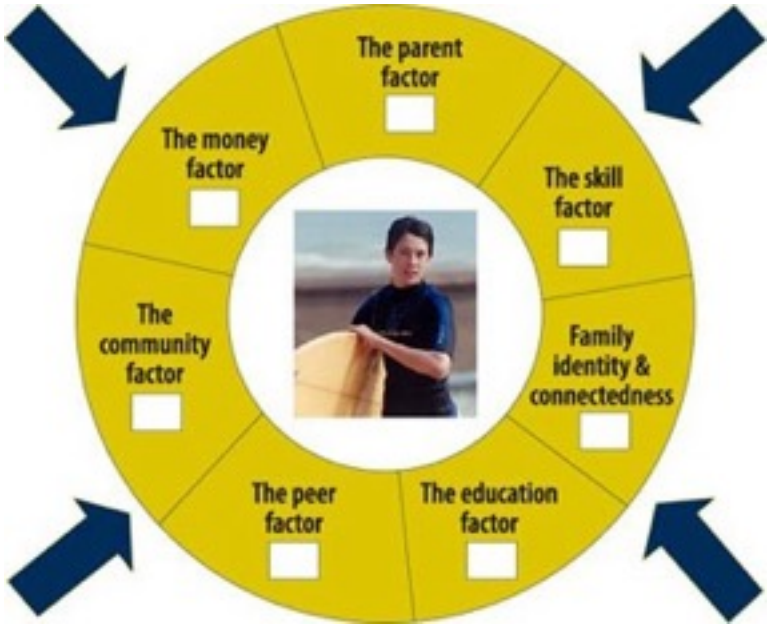
Gist: 'Flexibility' concerns the
'in-built' ability of the system
to adapt,
or to be adapted,
by its users,
to suit conditions
(without any fundamental system modification
by system development).

Type: Complex Quality Requirement.

Includes: {Connectability, Tailorability}.

See next 2 slides!

Possible Synonyms: Resilience, Robustness

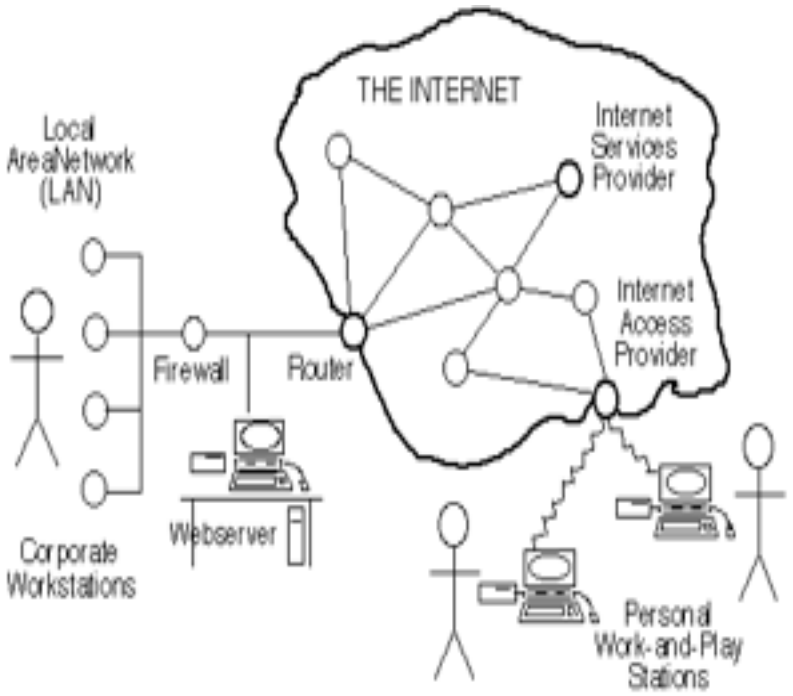


Connectability:

'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].

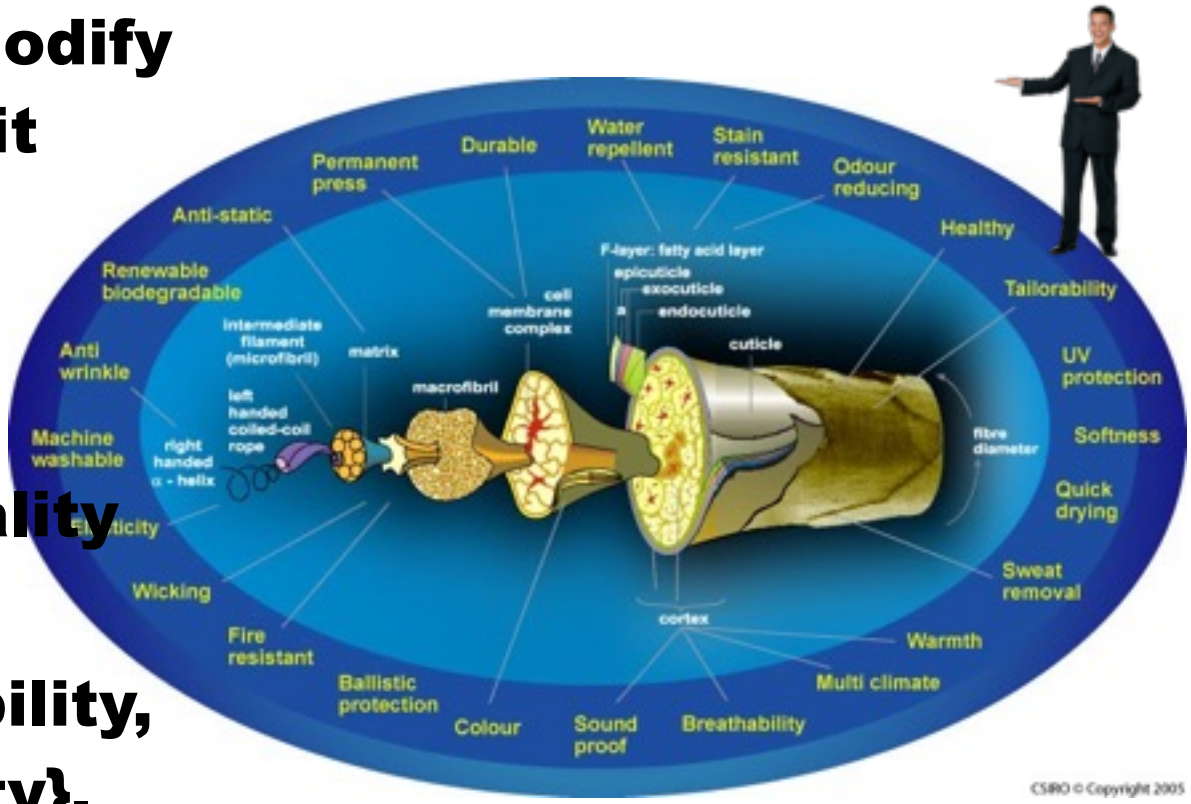
Tailorability:

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

Part Of: Flexibility.

Type: Complex Quality Requirement.

Includes: {Extendibility, Interchangeability}.



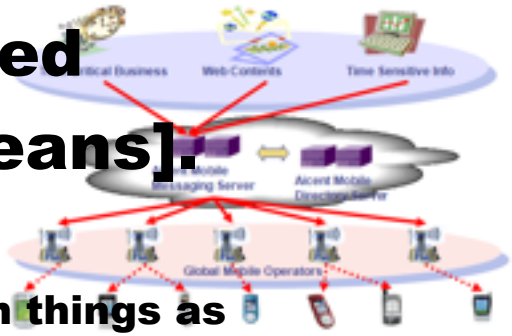
Multiple Attributes of Wool Fiber !

Extendibility:

Part Of: Tailorability.

Synonym: Scalability.

using a defined [Extension Means].



Scale: The cost to add

to

a defined [System]

a defined [Extension Class]

and defined [Extension Quantity]

"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 1/2:
'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}.

Upgradeability 2/2 :

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

- **Portability:**
- **Gist: 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 a system.**
- **Installability**
- **Gist: The ability to replace system components with others, which possesses improved (function, performance, cost and/or design) attributes. 'The cost to install the system in defined conditions.'**
- **Scale: The cost to add to a defined [System] a defined [Improvement] using a defined [Means].**

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 of performance.**
- **to then discover appropriate architecture,**
  - **capable of delivering those levels of performance, hopefully within resource constraints, and**
- **define the architecture in such detail , that we can**

estimate the probable impact of the architecture,

  - **on the requirements (Impact Estimation) so that the intent cannot be misunderstood by implementers,**
  - **and the desired effects are bound to be delivered.**
- **then later, 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.**

Architecture Level Impact Estimation Table

You can consider Maintainability together with other objectives!

	.....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

Maintainability Engineering Sources

- Gilb ACCU 'Designing Maintainability'
  - 90 minutes Slides.ppt (10.41 Mb)
  - http://www.gilb.com/tiki-download_file.php?fileId=171
- Maintainability Paper
  - http://www.gilb.com/tiki-download_file.php?fileId=138
- Competitive Engineering, Chapter 5, Scales of Measure
  - http://www.gilb.com/tiki-download_file.php?fileId=26

DZIĘKUJĘ ZA UWAGĘ - thank you for your attention
Next slide special offer!

Ask me for free digital copy!
(tom@gilb.com) Subject: 'BOOK'
The Architecture Language: 'Planguage'

I will then also send you, slides link, recent papers

You can view papers, these slides ("Super Methods") and 2 chapters of the CE book at www.GILB.com/downloads

Bio Gilb

- Tom Gilb was born in California in 1940, but 'escaped' to Europe at 15. He joined IBM Norway in 1958, and started his consultancy in 1960. He has published 9 Books, including 'Principles of Software Engineering' (1988, 20th Printing now, cited as major Agile inspiration), 'Software Inspection' (14th Printing) and Competitive Engineering (2005, 3rd Printing). He is a frequent contributor to the Polish 'Core Magazine' (http://www.coremag.eu/) . He consults with CIO/CTO/CEO level management worldwide on software and systems productivity and quality. More at www.Gilb.com