"The Ten Most Powerful Principles for Quality in [Software and] Software Organizations"

By
Tom Gilb

ITSF Oct 24 2000

ABSTRACT

Software knows it has a problem. Solutions abound.

But which solutions work?

What are the most fundamental underlying principles we can observe behind those successful solutions?

Can these principles guide us to select successful solutions and avoid time wasters?

One hint: in observing successful software organizations in the US, the dominant principle seems to be feedback and control.

TomGilb@Result-Planning.com
Version 0.9 July 4 2000

1. Feedback

Rapid feedback allows rapid correction.

The single most powerful principle in practice for software engineering is 'Feedback': facts about how things are working. The presumption is that the feedback is early enough to do some good. We have to have the time to make use of the feedback in order to change direction, if necessary.
All the other principles in this paper are really feedback-supporting ideas to help you get better control of your project.
Methods using rapid feedback succeed, those without seem to fail.
I have arrived at this conclusion by observing, for 42 working years, the principles which both succeed for me and for others.
Experience of formal feedback methods is decades old, and many sources do appreciate the power of these software feedback methods. But, far too many software engineers and their managers do not seem to have any real understanding or appreciation of these methods, and are still practicing low-feedback methods such as reviews, Quality Function Deployment (QFD, see ref. Akao90) testing, and Waterfall (also known as Big Bang, Grand Design) project management. Far too many textbooks continue to present the low-feedback methods; not from conscious rejection of high-feedback methods, but from ignorance of well-documented and decades-old experiences.

Methods:
Here are some of the most notable high-feedback methods.

Defect Prevention Process or 'DPP' (equals Software Engineering Institute CMM Level 5, see MAYS95, practiced at IBM from 1983-1985 and on).

The Defect Prevention Process is a successful way to remove the root causes of defects. In the short term (1 year) about 50% defect reduction is experienced, 70% reduction (compared with previously) is expected within 2-3 years and over 95% defect reduction in the 5-8 year timeframe. (Sources IBM Experience, Raytheon Experience [DION95]). The key feedback idea, which works here, compared to the 10 year earlier IBM failed attempts [FAGAN76], is to 'decentralize' the initial defect causal analysis activity to the grass roots programmers and analysts. This gives you true causes and realistic acceptable change suggestions. Then, the deeper 'cause analysis' and 'measured process-correction' work is undertaken by specialized Process Improvement Teams, outside of deadline-driven projects.
The feedback mechanisms are many; such as same-day feedback from the people working with the specification, and early numeric process change-result feedback from Process Improvement Teams.

Inspection (Fagan, IBM 1975)

The specification Inspection method, was originated by IBM (M. Fagan, H. Mills ('Cleanroom'), and R. Radice, CMM inventor).
Inspection has changed character since then. It was primarily focussed on bug removal in code and code design documents. Today it can more cost-effectively be used to measure Major Defect (software standards violations) level, using sampling, of any software or upstream marketing specification

[Gilb93]. The defect level measurement should be used to decide whether the entire specification can be released (exited) downstream. For example for a 'go:no-go' decision-making review or for further refinement (test planning, design, coding).
The main Inspection feedback components are:
• feedback to author from colleagues regarding software standards compliance.
• feedback to author about required levels of standards compliance in order to consider their work releasable.

Evolutionary Project Management (Mills, IBM, Cleanroom, 1970).
Evolutionary Project Management (Evo) [Mays96, Cotton96, Gilb88, Gilb00] has been successfully used on the most demanding space and military projects since 1970. The US Department of Defense changed their software engineering standard (2167a) to an Evo standard (MIL-STD-498 and then to succeeding public standards (example IEEE) standards derived from it).
The reports (op. cit.), and my own experience, is that Evo results in a remarkable ability to delivery on time and to budget, or better, compared to conventional project management methods [Morris94].
Typically an Evo project is consciously divided up into small, early, frequent stakeholder result-delivery steps. Each one cumulates towards satisfaction of final requirements. Step size is typically weekly or 2% of total time or budget. This results in excellent regular and realistic feedback about the team's ability to deliver meaningful measurable results to selected stakeholders. The feedback includes information on design suitability, stakeholders reaction, requirements tradeoffs, cost estimation, time estimation, people resource estimation, and development process aspects.

Statistical Process Control (SPC): Shewhart, Deming, Juran: (from 1920's) [Deming86].
SPC, although widely used in manufacturing, is only to a limited degree actually used directly in software work. But some use of it will be found in advanced uses of Inspection (Raytheon95, SEI97). The Plan Do Study (or Check) Act cycle is widely appreciated as a fundamental feedback mechanism.

2. Critical Measurement

If you do not focus on the few measures critical to your system, then it will fail.

It is true of any system that there are several factors which can cause them to die. It is true of your body, your organization, your project, and your software or service product. Doctors call such factors ‘critical factors’, and managers call them ‘Critical Success Factors’.

We know that far too many software projects fail totally, and almost all of them have some degree of failure, compared to initial plans and promises. The US Department of Defense estimated that about half of their software projects were total failures! (Source N Brown). The civil sector is no better [Morris95].

If you analyze the critical factors which caused the failure or the disappointments, you would get a list of factors to control better. They would include both qualitative stakeholder values (like serviceability, reliability, adaptability, portability, usability) and critical resources needed to deliver those stakeholder values (like people, time, money and data quality).

You would find, for these critical factors, a series of failures to manage more reasonably such as:

- failure to contract measurably for the critical factor
- failure to define the factor measurably
- failure to define a practical way to measure the factor
- failure to define the factor in any clear way whatsoever (buzzwords only)
- failure to design towards reaching that factors critical levels
- failure to maintain critical levels of performance during heavy loads or with changes to the system
- failure to make the entire project team aware of the numeric levels needed for the critical factors.
- failure to systematically identify all critical stakeholders and their critical needs.

The critical factors start with the ‘top ten’ (most influential ones) and can in practice be many more.

Our entire culture and literature of ‘software requirements’ systematically fails to account for most of these critical factors (while usually accounting for a few of them, such as performance, budget and deadline). Most of the quality factors are not defined quantitatively at all. They can in practice always be defined with a useful scale of measure. But, people are not trained to do this and their managers are no exception. Our ability to define critical ‘breakdown’ levels of performance, and to manage successful delivery is destroyed from the outset.

3. Multiple Objectives

If you cannot control multiple measures of quality and cost simultaneously, then your system will fail due to the ones you did not control.

You cannot manage just a few critical software project factors, and avoid dealing with others. You must simultaneously manage all the critical factors. If not then the 'forgotten factors' will probably be the very reasons for project or system failure. You do not have the luxury of managing qualities and costs which you are most familiar with. You have to deal with all the potential threats to your project, organization or system.

	Step #1 Plan A: {Design-X, Function-Y}	Step #1 Actual	Differe-nce. - is bad + is good	Total Step 1	Step #2 Plan B: {Design Z, Design F}	Step #2 Actual	Step #2 Differe-nce	Total Step 1+2	Step #3 Next step plan
Reliabil-ity 99%-99.9%	50% ±50%	40%	-10%	40%	30% ±20%	20%	-10%	60%	0%
Perform-ance 11sec.-1 sec.	80% ±40%	40%	-40	40	30% ±50%	30%	0	70%	30%
Usability 30 min. -30 sec.	10% ±20%	12%	+2%	12%	20% ±15%	5%	-15%	17%	83%
Capital Cost 1 mill.	20% ±1%	10%	+10%	10%	5% ±2%	10%	-5%	20%	5%
Enginee-ring Hours 10,000	2% ±1%	4%	-2%	4%	10% ±2.5%	3%	+7%	7%	5%
Calend-ar Time	1 week	2 weeks	-1week	2 weeks	1 week	0.5 weeks	+0.5 wk	2.5 weeks	1 week

Most software engineers have not yet learned to specify ***all*** their critical factors ***quantitatively***. So the ***next*** step, tracking progress against these goals, becomes impossible, to begin with.

The 'Impact Estimation Table, above, is conceptually similar to Quality Function Deployment tables [Akao90] but it is much more objective and numeric; and adaptable to other tasks such as (here) project tracking. It gives a better checkable picture of reality. [Gilb88, Gilb00]. The underlying detail is not visible here, but the % estimates are based on source-quoted, credibility evaluated, objective documented evidence, as far as possible.

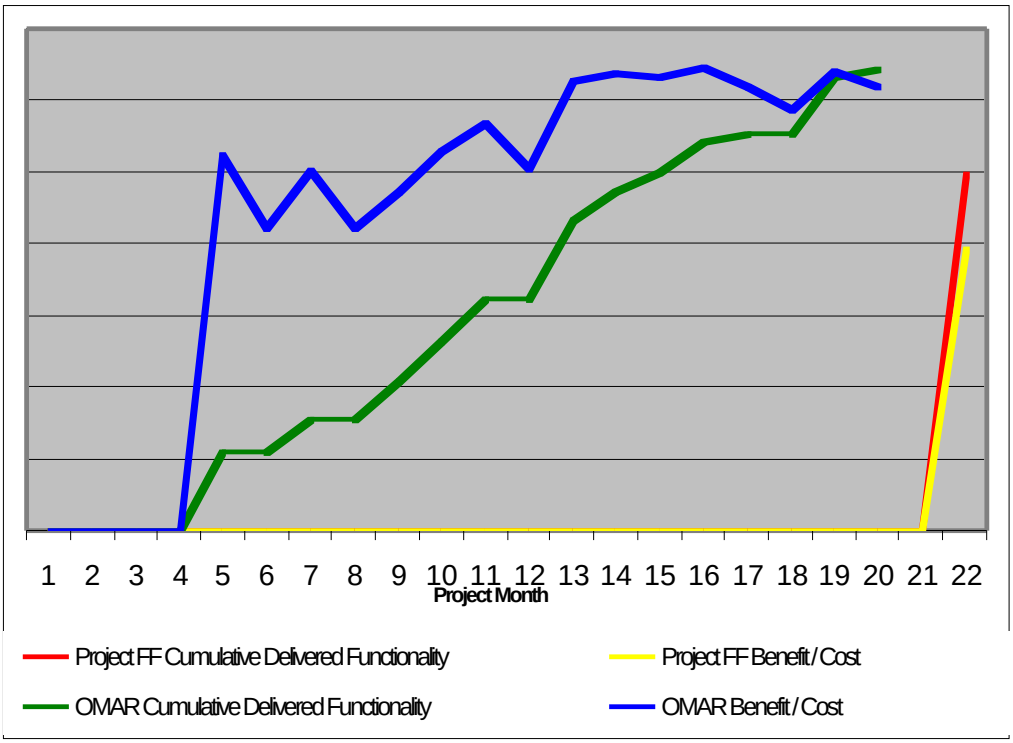
The impact estimation table can be used to evaluate ideas ***before*** their application, and it can also be used (as in the example above) to track progress towards multiple objectives ***during*** an Evolutionary project.

In this case the 'Actual' and 'Difference" and 'Total' numbers represent ***feedback*** in small steps for any chosen set of critical factors management wishes to monitor. If the project is deviating from plans, this will be easily visible and can be corrected on the next step.

4. Evolution

You must evolve in small steps towards your goals; large step failure kills the entire effort. And early frequent result delivery is politically and economically wise. 2% of total is a small step, that you can afford to fail on

Software engineering is by nature playing with the unknown. If we already had exactly what we needed, the reproduction cost of software dictates re-use of existing software. When we choose to develop software, there are many types of unknowns which threaten the result. One way to deal with these many unknowns is to tackle them in small increments, one at a time. If something goes wrong, we will immediately know it. We can also retreat to the previous level of satisfactory quality, until we understand how to progress again.



From Woodward99: one advantage of Evo is that you can focus on delivering high value increments to critical stakeholders early. The upper line represents high value at early stages.

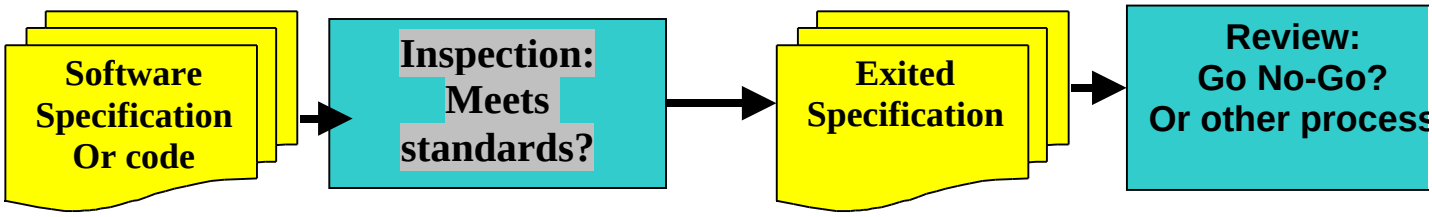
It is important to note that the small increments are not mere development increments. The point is that they are incremental satisfaction of identified stakeholder requirements. Early stakeholders might be salespeople needing a working system to demonstrate; system installers/help desk/service/testers who need to work with something, and finally, early trial users.

The small increments are typically a week or so. The smallest widely reported increments are the daily builds used at Microsoft, which are useful-quality systems, and their 6-10 week ‘shippable quality’ milestones [CUSOMANO95].

5. Quality Control

Quality Control must be done as early as possible, in planning, to reduce the delays from late defect finding.

Use numeric Exit from development process
Like “Maximum 0.2 Majors/Page”
Use Inspection sampling to keep costs down, and to permit early, before completion, action and learning.



There needs to be strong specification standards (like: ‘all quality requirements must be quantified’) and rigorous checking to measure that the rules are applied in practice. When the specs are not of some minimum standard (like < 1 Major defect/page remaining) then they must be edited until they become acceptable.

It is important that quality control be done very early in large works-in-progress, for example within the first 10 pages of work. If the work is not up to standard then the process can be corrected before more effort is wasted. I have seen (1986, Sweden) 40,000 pages Air traffic control logic design get ‘approved’ by 7 managers for coding. But those same managers showed later in a half day of real inspection, that I led them through, that there were about 19 logic defects per page, based on a random sample of 3 pages! Needless to say they were seriously late.

In another case I facilitated (USA, 1999, Jet parts supplier) 8 managers sampled 2 pages of 82 requirements pages and measured that there were 150 ‘Major’ defects per full page. Unfortunately they had failed to do such sampling 3 years earlier when their project started, so they had already experienced one year of delay; and told me they expected another year delay while removing injected defects from the project. This 2 year delay was accurately predicted from the defect density they found, and the known loss from Major defects. They were amazed at this insight, but agreed with the facts. In theory they could have saved 2 project years by doing early proper quality control against simple standards (clarity, unambiguous, and not design in requirements were all we used) as the requirements were being written, or at least before they were used to manage the project.

These are not unusual cases. I find them on a weekly basis all over the world. Management allows really bad specifications to go unchecked into costly project processes. They are obviously not managing properly.

6. Motivation

The ‘best methods’ work only when people are motivated. ‘Drive out fear’ (Deming86)

Motivation is everything! When individuals and groups are not motivated positively. They will not move forward. When they are negatively motivated (fear, distrust, suspicious) they will resist change to new and better methods.

Motivation is itself a method type. In fact there are a lot of large and small contributions to your group’s ‘sum of motivation’.

We can divide the ‘motivation problem’ into four useful categories:
• the ***will*** to change
• the ***ability*** to change
• the ***knowledge*** of change direction
• the ***feedback*** about progress in the desired change direction.

Leaders (I did not say ‘managers’) create the will to change by giving people a positive, fun, challenge, and the freedom and resources to succeed.

John Young, CEO of Hewlett Packard during the 1980’s, inspired his troops by saying that he thought they needed to aim to be ***measurably*** ten times better in service and product qualities (“10X”) by the end of the decade (1980-1989). He did not demand it. He ***supported*** them in doing it. They failed. Slightly! They reported getting about 9.95 times better, on average, in the decade. The company was healthy and competitive, during a terrible time for many others, such as IBM.

The knowledge of change ***direction*** is critical, otherwise energy moves people in the ‘wrong’ directions. In the software and systems world, this problem has three elements:
• ***measurable quantified clarity*** of variable stakeholder requirements and objectives
• knowledge of ***all the critical multiple*** simultaneous goals to move towards
• ***formal awareness of constraints***, while moving towards goals; such as resources and laws.

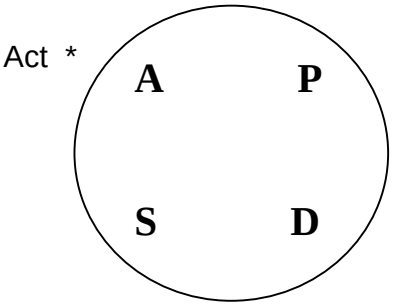
These elements are dealt with in other principles here, but they are a constant problem, because:
• we do not systematically convert our ‘change directions’ into crystal clear measurable ideas; thus permitting numeric feedback about movement in the ‘right’ direction.
We are likely to say we need a ‘robust’ or ‘secure’ system; and less likely to convert these rough ideals into concrete measurable defined agreed requirements or objectives.

• we focus too often on a single measurable factor (‘% built’) when our reality demands that we simultaneously track multiple critical factors to avoid failure and to ensure success***. We don’t get enough ‘rich’ feedback.***

7. Process Improvement

Eternal Process improvement is necessary as long as you are in competition (Paraphrasing Deming about PDSA cycle end).

"The Shewhart Cycle for Learning and Improvement
The P D S A Cycle



Act. Adopt the change, or Abandon it, or Run through the cycle again, possibly under different conditions. "

Exact reproduction (- '(Do)' from a letter to Tom Gilb from W. Edwards Deming 18 May, 1991

Our conventional project management ideas strongly suggest that projects have a clear beginning and a clear end. In our competitive world, this is not as wise a philosophy as the one Deming suggests. We can have an infinite set of 'milestones', such as Evolutionary steps of result delivery, as we need. But the moment we abandon the project; we hand opportunity to our competitors and enemies. They can sail past our levels of performance and take our markets.

The practical consequence is that our entire mind set must always set new ambitious numeric 'stakeholder value' targets, for our organizational capability and for our product and service capabilities. Projects must become ***eternal*** battles to get ahead and stay ahead.

Continuous improvement efforts in the software and services area at IBM, Raytheon and others [Mays95, Dion95, Kaplan94, HP (10X, Young)] show that we can improve critical cost and performance factors by 20 to 1 in 5 to 8 year time frames. If we do not continually persist in doing so, our competition gets a free shot at us.

8. Persistence

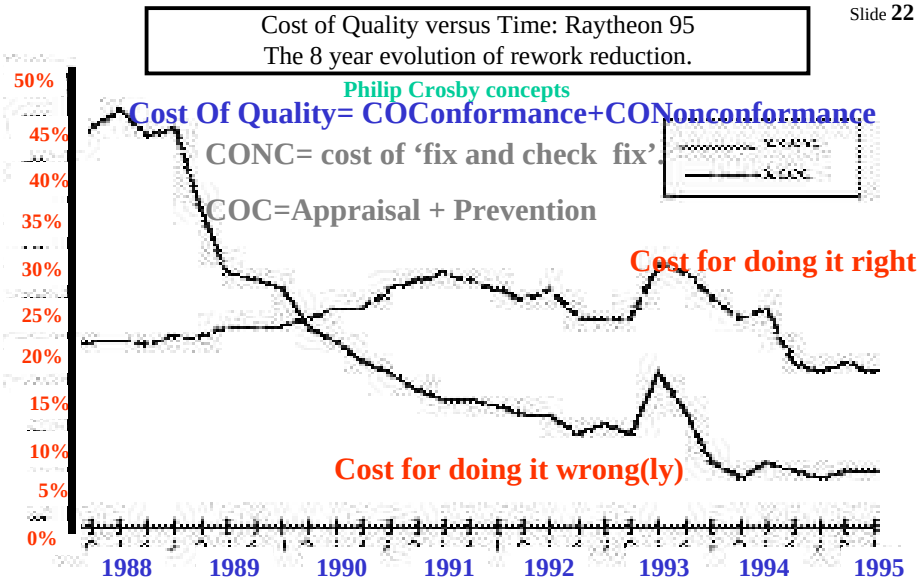
Years of persistence are necessary to change a culture.

W. Edwards Deming
"It takes 2-3 years to change a project, and a generation to change a culture."

Piet Hein (Denmark)
"Things Take Time" (TTT)

"Despite mistakes, disasters, failures, and disappointment, Leonardo never stopped learning, exploring, and experimenting. He demonstrated Herculean persistence in his quest for knowledge. Next to a drawing of a plow in his notebook Leonardo proclaimed, "I do not depart from my furrow." Elsewhere he noted, "Obstacles do not bend me" and "Every obstacle is destroyed through rigor." [GELB98, p79].

Slide **22**



Project Cost = {Cost of Quality + Cost of Performance}.
Cost of Performance={Planning, Documentation, Specification}.

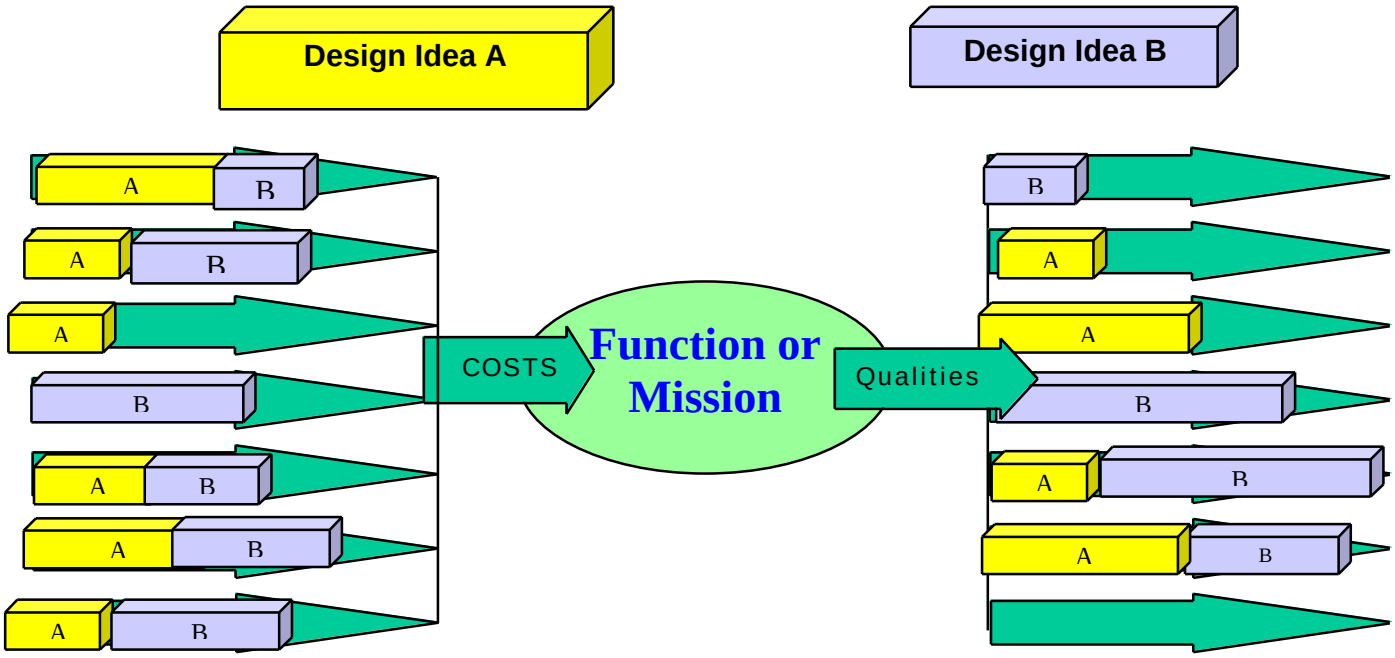
In the case of Raytheon process improvements [DION95] many years of persistent process change, for 1,000 programmers, was necessary to drop rework costs from 43% of total software development costs, to below 5%.

Technical management needs to have a long term plan for improvement of the critical characteristics of their organization and their products. Such long term plans need to be numerically trackable, and to be stated in multiple critical dimensions. At the same time visible short term progress towards those long term goals should be planned, expected and tracked.

9. Multiple Impacts

Any method (means, solution, design) you choose will have multiple quality and cost impacts, whether you like them or not!

We need to estimate all impacts on our objectives.
We need to reduce, avoid or accept negative impacts.
We must avoid simplistic one-dimensional arguments.



A and B are solutions/strategies/designs, the 'means' by which we intend to satisfy the 'quality' (stakeholder values) requirements. They each have an effect on many of the quality requirements, and on many of the cost budgets. The length of the bar indicates the degree of impact towards the goal or budget level (symbolized by the arrow point).

In order to get a correct picture of how good any idea is, for our purposes, we must

• have a quantified multidimensional specification of our quality objectives (ends, values, good stuff) and of our budget-limited resources (people, time, money). = Principle 3 above.
• have knowledge of the expected impact of each 'means' on all the 'ends' (qualities) and 'costs'
• evaluate an idea with respect to its total, expected or real, impact on our residual (un-met) objectives and un-used cost budgets.

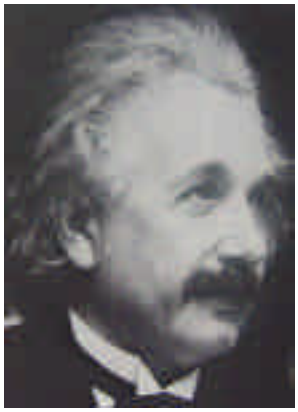
If we fail to use this systems engineering discipline, then we will be met with the unpleasant surprises of delays, and bad quality, which seem to be the norm in software engineering today.

One practical way to model these impacts is using an impact estimation table (as in the example from Principle 3 above).

10. Results Orientation

You must keep your focus on the essential results, and never fall victim to the means.

"Perfection of means and confusion of ends seem to characterize our age"



Albert Einstein.

The problems with software requirements.
One of our most common problems is that we seem unable to specify what we really want! We have many problems there:
• we specify the 'means' not our true 'ends'
• we specify unclearly (not testable, not measurably)
• we totally fail to identify key stakeholders and their needs
• we ignore specification of constraints, assumptions
• we fail to specify key quality goals at all (serviceability, recoverability,…)
• we fail to specify key economic constraints (future maintenance cost …)

Our currently published requirements specification and analysis discipline is suited only for a program coder ('functional requirements'), but not for a software engineer or systems engineer.

To discover the problem we have only to ask of a specification: "Why?", and the answer will be a higher level of specification, nearer the real ends. There are too many designs in our requirements!

You might say, why bother? Isn't the whole point of software to get the code written? Who needs high level abstractions; cut the code! But somehow that code is late and of unsatisfactory quality. And the reason is, partly, lack of attention to the real needs of the stakeholders and the project. We need these high-level abstractions of what our stakeholders need ***so that we can focus on giving what they are paying us for!*** So that we can design to find the best technology to satisfy their needs at a competitive cost.

One day software engineers will realize that the primary task is to satisfy their stakeholders. They will learn to design towards stakeholder requirements, which are multiple simultaneous requirements. One day we will become real systems engineers and realize that there is far more to software engineering than writing code!

The summary principle

Motivate people towards real results by giving them numeric feedback frequently and the ability to change anything for success.

It is that simple to specify. And, it is that difficult to do.

References

Akao90: Yoji Akao (Editor), **"QUALITY FUNCTION DEPLOYMENT: Integrating Customer Requirements into Product Design",** Productivity Press, Cambridge Mass. USA, 1990.

COTTON96: Cotton, Todd, "**Evolutionary Fusion: A Customer-Oriented Incremental Life Cycle for Fusion".** Hewlett-Packard Journal, August 1996, Vol. 47, No. 4, pages 25-38. This is adapted from the book Object-oriented Development at Work: Fusion in the Real World, by Ruth Malan et al (Eds.), Prentice Hall PTR, 1996.
An excellent detailed view of the Evolutionary project management process as taught Corporate-Wide at HP. The author (Todd Cotton) was on a project team at HP in 1989 which Gilb taught early versions of the Planguage method. See another member of that Project in MAY96.

CUSUMANO95 : Michael A. Cusumano and Richard W. Selby.**: "Microsoft Secrets : How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People",** The Free Press (div. of Simon and Schuster), 1995,
ISBN 0-02-874048-3, 512 pp.

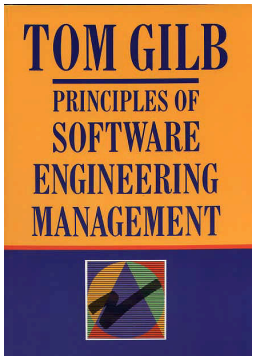
DEMING86: Deming, W. Edwards, **Out of the Crisis**, MIT CAES Center for Advanced Engineering Study, Cambridge MA USA-02139, 1986, ISBN 0-911379-01-0, 507 pages , hard cover.

DION95: Raymond Dion, "The Raytheon Report", http://www.sei.cmu.edu/products/publications/95.reports/95.tr.017.html. This is an update and detail of DION93. Over 80 pages. It is highly recommended as a study of quantified process improvements.

FAGAN76: Fagan, M. E. 1976. Design and code inspections, IBM Systems Journal 15, (3) 182-211. Reprinted IBM Systems Journal, Vol. 38, Nos. 2&#, 1999, pp. 259-287
www.almaden.ibm.com/journal

GELB98: Michael J. Gelb, How to Think Like Leonardo da Vinci, Dell Publishing NY, 1998, ISBN 0-440-50827-4, Feb 2000 Edition Paperback. Author email:DaVincian@AOL.com

GILB88: Tom Gilb**, "Principles of Software Engineering Management",** Addison-Wesley, UK/USA, 1988. 442 pages. ISBN 0-201-19246-2. Soft-cover. £24.95.



GILB93: Gilb, T. and Graham, D, **"Software Inspection"**, Addison-Wesley, 1993 471 pp. ISBN 0-201-63181-4. For complete corresponding Inspection (DQC) course materials see [GILB-WWW].
Japanese Translation, August 1999, Yen 5,000
ISBN 4-320-09727-0, C3041 (code next to ISBN)
450 pages including index, softcover
akagi@kyoritsu-pub.co.jp

Gilb00: Gilb, Tom
Competitive Engineering, Addison-Wesley, UK, End 2000.
There will always be a version of this book free at my website www.result-planning.com.

KAPLAN94: Kaplan, Craig, Clark, Ralph and Tang, Victor, **Secrets of Software Quality, 40 Innovations from IBM,** McGraw Hill., ISBN 0-07-911975-3, 383 pages.
Author email: ckaplan@iqco.com (Craig)

KEENEY92, Keeney, Ralph L., "**Value-focused Thinking: A Path to Creative Decisionmaking"** (sic.)
Paperback / Published 1996
www.Amazon.com Price: $18.95

(Note Hardback edition is out of print: Harvard University Press, Cambridge Mass/London 1992, ISBN 0-674-93197-1. $37.50.)

MAY96: Elaine L. May and Barbara A. Zimmer, "**The Evolutionary Development Model for Software",** Hewlett-Packard Journal, August 1996, Vol. 47, No. 4, pages 39-45.
This is an excellent complimentary article to COTTON96. Elaine attended a Gilb course at HP in 1989.

MAYS95: Robert Mays (slides in 12th Intl. Conference and Expo on Testing Computer Software)
- June 12-15 1995, his lecture June 13th, Wash DC.

- **“IBM Defect Prevention Process and Test”**
- ***Note he has a full bibliography of his writings(such a central IBM SJ No. One 1990) in this (6 items incl. his chapter in Gilb & Graham 'Software Inspection' book, GILB93:Chapter 17)***
-

Author Contact: Robert Mays 1995-
- P O Box 12195, Dept. A82, Bldg. B503, RTP NC 27709
Telephone: (919) 254 5210
Email: rmays@us.ibm.com

MORRIS94: Morris, Peter W G, **THE MANAGEMENT OF PROJECTS**, Thomas Telford, London, 1994, 358 pages, ISBN 0 7277 1693 X, £55, Publisher 1 Heron Quay, London E14 4JD. Tel. 0171-987-6999, Fax 538-4101.

Peters00: Tom Peters,
Reinventing Work, the project 50. Alfred A. Knopf, New York, 2000, ISBN 0-375-40773-1. See Peters’ website www.tompeters.com, $15.95
See also his book ‘the Quick Prototype50’.
See especially his emphasis on ‘quick prototyping’ in relation to Evolutionary project management.

SEI97: William A. Florac, Robert E. Park, Anita D. Carleton, **Practical Software Measurement: Measuring for Process Management and Improvement.**
Guidebook from Software Engineering Institute. Reference: CMU/SEI-97-HB-003.
Downloadable from SEI web site (Acrobat Reader): ftp://ftp.sei.cmu.edu/ for publications, and main site http://www.sei.cmu.edu. April 1997, 240 pages
This excellent free publication is based on the principles of statistical process control taught by Shewhart and Deming. It is specifically oriented towards the software community. It has exceptionally good advice on how to determine process and product metrics.
Contains an Appendix by Mark Roberts of McDonnell Douglas about use of SPC methods on Software Inspection (DQC).
Email: florac@sei.cmu.edu (William Florac),
rep@sei.cmu.edu (Robert Park),
adc@sei.cmu.edu (Anita Carleton)

Woodward99: Stuart Woodward: “Evolutionary project Management”
IEEE Computer Oct 1999, page 49-57
s.woodward@computer.org