Nov 2002 Issue

CROSSTALK
The Journal of Defense Software Engineering

The 10 Most Powerful Principles for Quality in Software and Software Organizations

Tom Gilb, Result Planning Limited

The software industry knows it has a problem: The industry's maturity level with respect to "numbers" is known to be poor. While solutions abound, knowing which solutions work is the big question. What are the most fundamental underlying principles in successful projects? What can be done right now? The first step is to recognize that all your quality requirements can and should be specified numerically. This does not mean "counting bugs." It means quantifying qualities such as security, portability, adaptability, maintainability, robustness, usability, reliability, and performance. This article presents 10 powerful principles to improve quality that are not widely taught or appreciated. They are based on ideas of measurement, quantification, and feedback.

All projects have some degree of failure compared with initial plans and promises. Far too many software projects fail totally. In the mid 1990s, the U.S. Department of Defense (DoD) estimated that about half of its software projects were total failures [1]. The civil sector is no better [2]. So what can be done to improve project success? This article outlines 10 key principles of successful software development methods that characterize best practices.

These 10 principles have been selected because there is practical experience showing that they really gain control over qualities and their costs. They have a real track record spanning decades of practice in companies like IBM, Hewlett Packard, and Raytheon. They are not new: They are classic. But the majority of our community is young and experientially new to the game, so my job is to remind the industry of the things that work well. Your job is to evaluate this information and start getting the improvements that your management wants in terms of quality and the time and effort needed to get them.

"Those who do not learn from history, are doomed to repeat it" [3].

Principle 1: Use Feedback

The practice of gaining experience from formal feedback methods is decades old, and many appreciate its power. However, far too many software engineers and their managers are still practicing low feedback methods, such as waterfall project management (also known as Big Bang or Grand Design). Even many textbooks and courses continue to present low feedback methods. This is not done in conscious rejection of high feedback methods but from ignorance of the many successful and well-documented projects that have detailed the value of high feedback methods.

Methods using feedback succeed; those without feedback seem to fail. Feedback is the single most powerful principle for software engineering. (Most of the other principles in this article support the use of feedback.) Feedback helps you get better control of your project by providing facts about how things are working in practice. Of course, the presumption is that the feedback comes early enough to do some good; rapid feedback is the crux. We need to have the project time to make use of the feedback (for example, to radically change direction, if that is necessary). Four of the most notable rapid high-feedback methods are discussed in the following sections:

Defect Prevention Process

The Defect Prevention Process (DPP) equates to the Software Engineering Institute's Capability Maturity Model® (CMM®) Level 5 as practiced at IBM from 1983 to the present [4]. The DPP is a successful way to remove the root causes of defects. In the short term (one year) about a 50 percent defect reduction can be expected; within two to three years, about a 70 percent reduction (compared to the original level) can be experienced; and in five to eight years, about a 95 percent defect reduction is possible [5].

The key feedback idea is to *decentralize* the initial causal analysis activity by investigating defects back to the grassroots programmers and analysts. This gives you the true causes and acceptable, realistic change suggestions. Deeper *cause analysis* and *measured process-correction* work can then be undertaken outside of deadline-driven projects by the more specialized and centralized process improvement teams.

There are many feedback mechanisms. For example, same-day feedback is obtained from the people working with the specification, and early numeric process change-result feedback is obtained from the process improvement teams.

Inspection Method

The Inspection Method originated at IBM in work carried out by M. Fagan, H. Mills (cleanroom method), and R. Radice (CMM inventor) [6]. Originally, it primarily focused on bug removal in code and code-design documents. Many continue to use it this way today. However, inspection has changed character in recent years. Today, it can be used more cost-effectively by focusing on measuring the significant defects on upstream specifications. Furthermore, sample areas often only need to be inspected rather than processing the entire document [7]. For example, the defect level measurement should be used to decide whether the entire specification is fit for release downstream to be used for a *go/no-go* decision-making review or for further refinement (test planning, design, or coding).

The main Inspection Method feedback components are as follows:

- Feedback to author from colleagues regarding compliance with software standards.
- Feedback to author about required levels of standards compliance in order to consider their work releasable.

Evolutionary Project Management

Evolutionary Project Management (Evo, which originated in large scale within cleanroom methods) has been successfully used on the most demanding space and military projects since 1970 [8, 9]. The DoD changed its software engineering standard MIL-STD-2167A to an Evo standard (MIL-STD-498), which derived succeeding public standards, (for example, the Institute of Electrical and Electronics Engineers). The reports, (op. cit.) along with my own experience, are that Evo results in a remarkable ability to deliver on time and on budget, or better, compared to conventional project management methods [2].

An Evo project is consciously divided into small, early, and frequently delivered stakeholder result-focused steps. Each step delivers benefits and builds toward satisfaction of the final requirements. Step size is typically weekly or 2 percent 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' reactions, requirements' trade-offs, cost estimation, time estimation, people resource estimation, and development process aspects.

Statistical Process Control

Statistical Process Control [10], although widely used in manufacturing [11], is only used in software work to a limited degree. Some use is found in advanced inspections [5, 12]. The Plan Do Study Act cycle is widely appreciated as a fundamental feedback mechanism.

Principle 2: Identify Critical Measures

It is true of any system - your body, an organization, a project, software, or service product - that there are several factors that can cause a system to die. Managers call these *critical success factors*. If you analyzed systems looking for all the critical factors that cause shortfalls or failures, you would get a list of factors needing better control. They would include both stakeholder values (such as serviceability, reliability, adaptability, portability, and usability) and the critical resources needed to deliver those values (i.e., people, time, money, and data quality). For each critical factor, you would find a series of faults that would include the following:

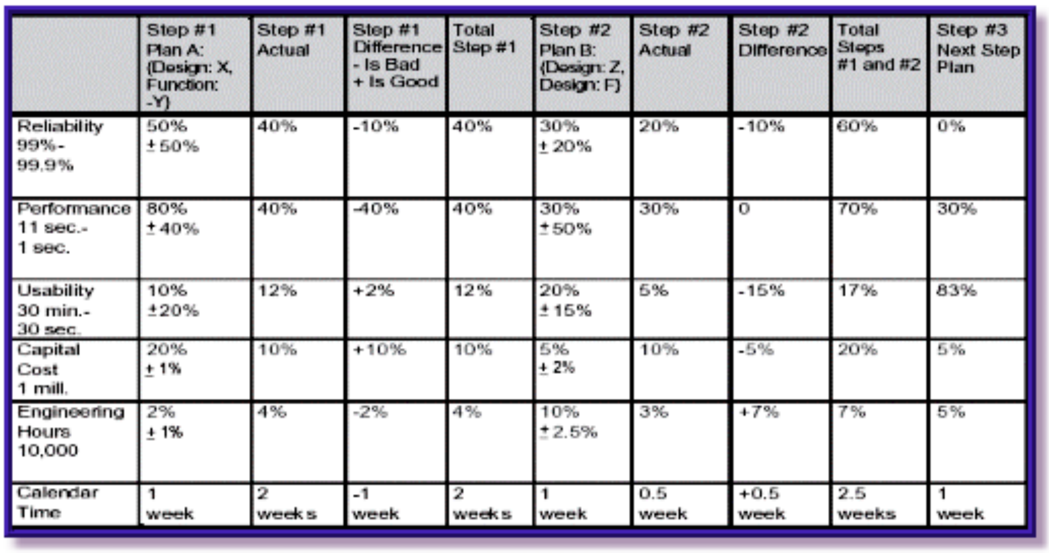
- Failure to systematically identify all critical stakeholders and their critical needs.
- Failure to define the factor measurably. Typically, only buzzwords are used and no indication is given of the survival (failure) and target (success) measures.
- Failure to define a practical way to measure the factor.
- Failure to contract measurably for the critical factor.
- Failure to design toward reaching the factor's critical levels.
- Failure to make the entire project team aware of the numeric levels needed for the critical factors.
- Failure to maintain critical levels of performance during peak loads or on system growth.

Our entire culture and literature of *software requirements* systematically fails to account for the majority of critical factors. Usually, only a handful such as performance, financial budget, and deadline dates are specified. Most quality factors are not defined quantitatively at all. In practice, all critical measures should always be defined with a useful scale of measure. However, people are not trained to do this and managers are no exception. The result is that our ability to define critical *breakdown* levels of performance and manage successful delivery is destroyed from the outset.

Principle 3: Control Multiple Objectives

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

I have developed the Impact Estimation (IE) method (see Table 1) to enable tracking of critical factors; however, it does require that critical objectives and quantitative goals have been identified and specified. Given that most software engineers have not yet learned to specify all their critical factors *quantitatively* (Principle 2), this *next* step, tracking progress against quantitative goals to enable control of multiple objectives (this principle), is usually impossible.



	Step #1 Plan A: (Design: X, Function: -Y)	Step #1 Actual	Step #1 Difference - Is Bad + Is Good	Total Step #1	Step #2 Plan B: (Design: Z, Design: F)	Step #2 Actual	Step #2 Difference	Total Steps #1 and #2	Step #3 Next Step Plan
Reliability 99%-99.9%	50% ±50%	40%	-10%	40%	30% ±20%	20%	-10%	60%	0%
Performance 11 sec.-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%
Engineering Hours 10,000	2% ±1%	4%	-2%	4%	10% ±2.5%	3%	+7%	7%	5%
Calendar Time	1 week	2 weeks	-1 week	2 weeks	1 week	0.5 week	+0.5 week	2.5 weeks	1 week

Table 1: *Example of an Impact Estimation Table*
(Click on image above to show full-size version in pop-up window.)

IE is conceptually similar to Quality Function Deployment [13], but it is much more objective and numeric. It gives a picture of reality that can be monitored [14, 15] (Table 1). It is beyond the scope of this article to provide all the underlying detail for IE. To give a brief outline, the percentage estimates in Table 1 are based, as far as possible, on source-quoted, credibility- evaluated, objective, documented evidence. IE can be used to evaluate ideas before their application, and it can also be used, as in Table 1, to track progress toward multiple objectives *during* an evolutionary project. In Table 1, the *Actual Difference* and *Total* numbers represent *feedback* in small steps for the chosen set of critical factors that management has decided to monitor. If the project is deviating from plans, this will be easily visible and can be corrected in the next step.

Principle 4: Evolve in Small Steps

Software engineering is by nature playing with the unknown. If we already had exactly what we needed, we would reuse it. When we choose to develop software, there are many types of risk that threaten the result. One way to deal with this is to tackle development in small steps, one step at a time. If something goes wrong, we will immediately know it. We also have the ability to retreat to the previous step, a level of satisfactory quality, until we understand how to progress again.

It is important to note that the small steps are not mere development increments. The point is that they incrementally satisfy identified stakeholder requirements (see Figure 1). Early stakeholders might be salespeople needing a working system for demonstration, system installers/help desk/service/testers who need to work with something, or early trial users.

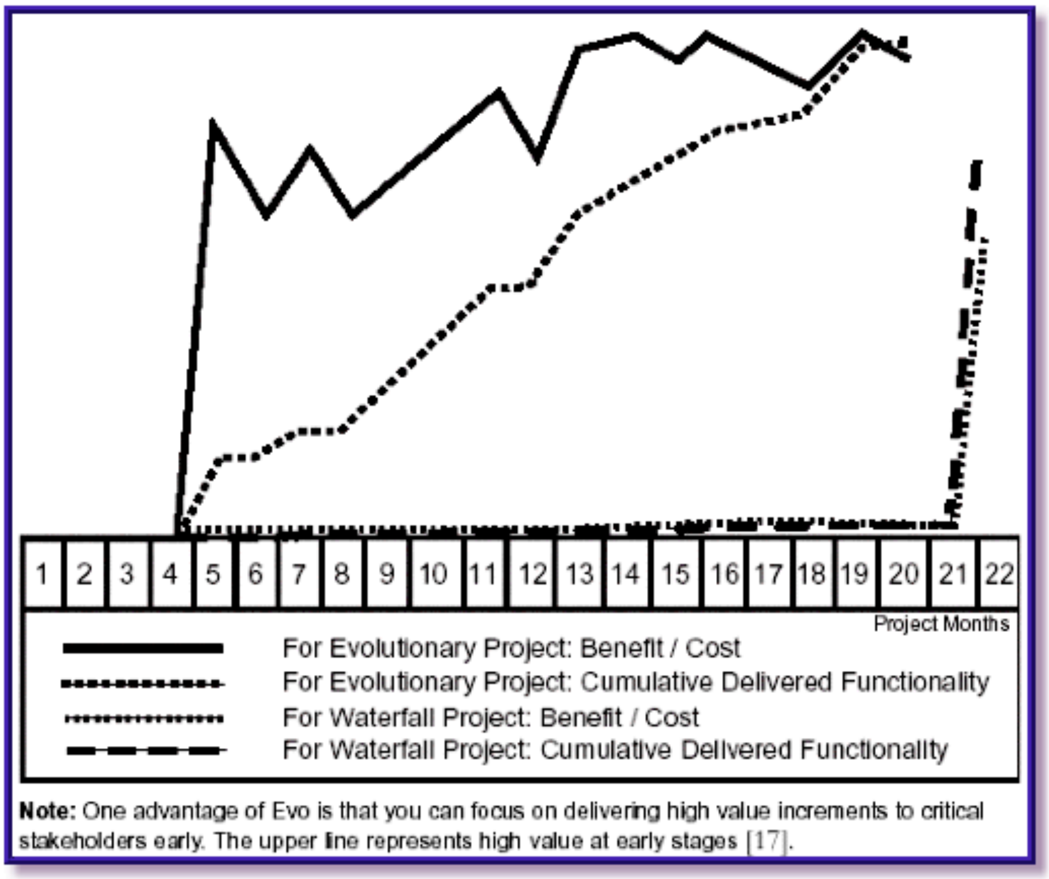


Figure 1: *Evolutionary vs. Waterfall Comparison*
(Click on image above to show full-size version in pop-up window.)

The duration of each small step is typically a week or so. The smallest widely reported steps are the daily builds used at Microsoft, which are useful-quality systems. They cumulate to six- to 10-week *shippable quality* milestones [16].

Principle 5:A Stitch in Time Saves Nine

Quality control must be done as early as possible, from the earliest planning stages, to reduce the delays caused by finding defects later. There needs to be strong specification standards (such as *all quality requirements must be quantified*) and rigorous checking to measure that the rules are applied in practice. When the specifications are not of some minimum standard (like ">1 major defect/page remaining") then they must be edited until they become acceptable, including the following:

- Use inspection sampling to keep costs down, and to permit early, i.e., before specification completion, correction and learning.
- Use numeric exit from development processes such as *Maximum 0.2 Majors per page*.

It is important that quality control by inspection be done very early for large specifications, 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 half a day of inspection (based on a random sample of three pages) show that there were about 19 logic defects per page in 40,000 pages of air traffic control logic design. The same managers who had originally *approved* the logic design for coding carried out the inspection with my help. Needless to say, the project was seriously late.

In another case I facilitated (United States, 1999, jet parts supplier), eight managers sampled two pages out of an 82-page requirements document and measured 150 *major* defects per page. Unfortunately, they had failed to do such sampling three years earlier when the project started, so they had already experienced one year of delay; they told me they expected another year delay while removing the injected defects from the project. This two-year delay was accurately predictable given the defect density they found and the known average cost from major defects. They were amazed at this insight, but agreed with the facts. In theory, they could have saved two project years by doing early quality control against simple standards: clarity, unambiguity, and no design in requirements.

These are not unusual cases. I find them consistently all over the world. Management frequently allows extremely weak specifications to go unchecked into costly project processes. They are obviously not managing properly.

Principle 6: Motivation Moves Mountains

Motivation is everything! When individuals and groups are not motivated positively, they will not move forward. When they are negatively motivated (fear, distrust, and suspicion), they will resist change to new and better methods. Motivation is a type of method. In fact, there are many large and small items contributing to your group's *sum of motivation*. We can usefully divide the *motivation problem* into four categories:

- The will to change.
- The knowledge to change direction.
- The ability to change.
- 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 and fun challenge and the freedom and resources to succeed. During the 1980s, John Young, CEO of Hewlett Packard, inspired his troops by saying that he thought they needed to aim to be measurably 10 times better in service and product qualities by the end of the decade. He did not demand it. He supported them in doing it. 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.

The knowledge of directional change is critical to motivation; people need to channel their energies in the right direction! In the software and systems world, this problem has three elements, two of which have been discussed in earlier principles. They are as follows:

- Measurable, quantified clarity of the requirements and objectives of the various stakeholders (Principle 2).
- Knowledge of all the multiple critical goals (Principle 3).
- Formal awareness of constraints such as resources and laws.

These elements are a constant communication problem because of the following:

- We do not systematically convert our directional changes into crystal clear measurable ideas; people are unclear about the goals and there is no ability to obtain 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-upon requirements or objectives.
- We focus too often on a single measurable factor (such as *percent built* or *budget spent*) when reality demands that we simultaneously track multiple critical factors to avoid failure and to ensure success. We do not understand what we should be tracking, and we do not get enough *rich* feedback.

Principle 7: Competition Is Eternal

Our conventional project management ideas strongly suggest that projects have a clear beginning and a clear ending. In our competitive world, this is not as wise a philosophy as one W. Edwards Deming suggests, "Eternal process improvement is necessary as long as you are in competition" [11]. We can have an infinite set of *milestones* or evolutionary steps of result delivery and use them as we need; the moment we abandon a project, we hand opportunity to our competitors. They can sail past our levels of performance and take our markets.

The practical consequence is that our entire mindset must always be on setting new ambitious numeric *stakeholder value* targets both for our organizational capability and our product and service capabilities (see Figure 2).

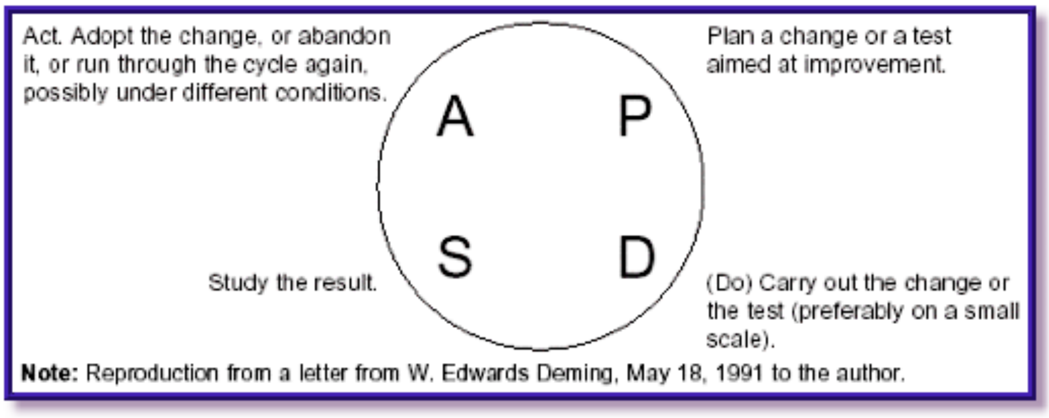

Act. Adopt the change, or abandon it, or run through the cycle again, possibly under different conditions.
Plan a change or a test aimed at improvement.
A
P
S
D
Study the result.
(Do) Carry out the change or the test (preferably on a small scale).
Note: Reproduction from a letter from W. Edwards Deming, May 18, 1991 to the author.

Figure 2: *The Shewhart Cycle for Learning and Improvement - the PDSA Cycle*

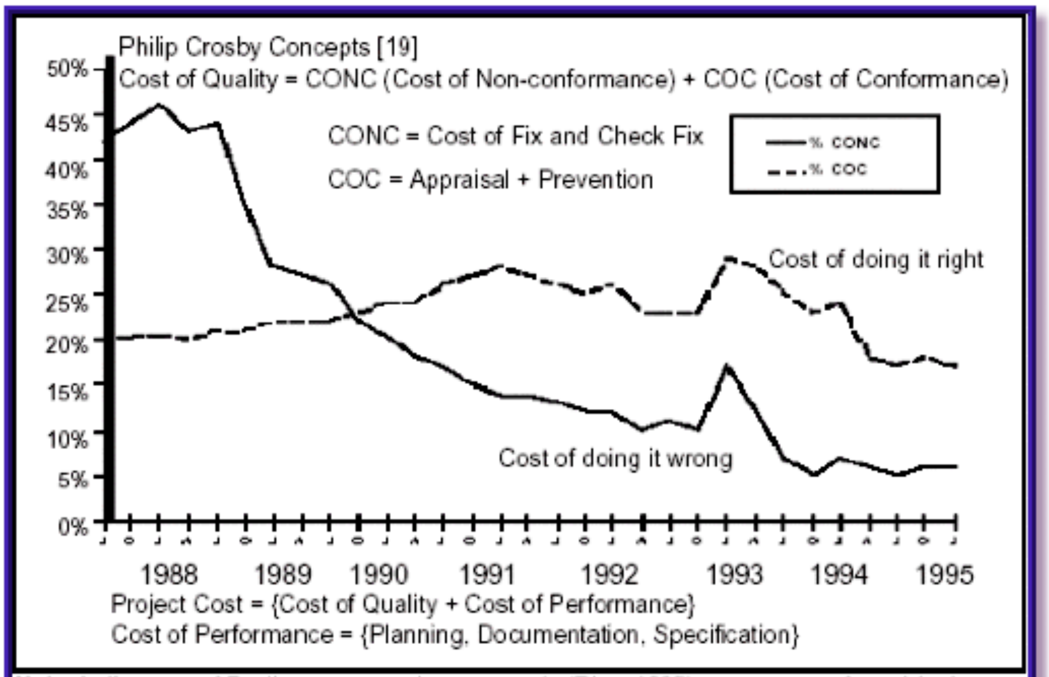
(Click on image above to show full-size version in pop-up window.)

Continuous improvement efforts in the software and services area at IBM, Raytheon, and others [4, 5, 18] show that we can improve critical cost and performance factors by 20 to one, in five- to eightyear time frames. Projects must become *eternal* campaigns to get and stay ahead.

Principle 8:Things Take Time

"It takes two to three years to change a project, and a generation to change a culture" [11].

Technical management needs to have a long-term plan for improving the critical characteristics of their organization and their products. Such long-term plans need the ability to be tracked numerically and stated in multiple critical dimensions. At the same time, visible short-term progress toward those long-term goals should be planned, expected, and tracked (see Figure 3).



Note: In the case of Raytheon process improvements (Dion, 1995), many years of persistent process change for 1,000 programmers were necessary to drop rework costs from 43% of total software development costs to below 5%.

Figure 3: *Cost of Quality vs. Time: Raytheon 95 - the Eight-Year Evolution of Rework Reduction*
(Click on image above to show full-size version in pop-up window.)

Principle 9:The Bad With the Good

Any method (means, solution, or design) you choose will have multiple quality and cost impacts whether you like them or not! In order to get a correct picture of how good any idea is for meeting our purposes, we must do the following:

- Have a quantified, multidimensional specification of our requirements, our quality objectives, and our resources (people, time, or money).
- Have knowledge of the expected impact of each design idea on all these quality objectives and resources.
- Evaluate each design idea with respect to its total - expected or real - impact on our requirements, the unmet objectives, and the unused cost budgets.

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. If we fail to use this systems engineering discipline, then we will be met with 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 IE table (see Table 1).

Principle 10: Keep Your Eyes on Where You Are Going

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

Einstein.

To discover the *real* problem, we have only to ask of a specification: Why? 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. The reason is often 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 them what they are paying us for! Our task is to design and deliver 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 toward stakeholder requirements (multiple simultaneous requirements). One day we will become real systems engineers and realize there is far more to software engineering than writing code.

Conclusion

Motivate people toward real results by giving them numeric feedback frequently and the freedom to use any solution that gives those results. It is that simple to specify. It is that difficult to do.

References

1. Jarzombek, Stanley J. "The 5th Annual Joint Aerospace Weapons Systems Support, Sensors, and Simulation Symposium (JAWS S3)." Proceedings, 1999.
2. Morris, Peter W. G. *The Management of Projects*. Ed. Thomas Telford. London, 1994.
3. Santayana, George. *The Life of Reason*. Amherst: Prometheus Books, 1903.
4. Mays, Robert. *Practical Aspects of the Defect Prevention Process*. (Gilb, Tom, and Dorothy Graham. *Software Inspection*. Addison-Wesley, 1993. Chapter 17 written by Mays).
5. Dion, Raymond, et. al. *The Raytheon Report*. Pittsburgh: Software Engineering Institute, 1995 www.sei.cmu.edu/publications/documents/95.reports/95.tr.017. html.
6. Fagan, Michael E. "Design and Code Inspections." *IBM Systems Journal* 15.3 (1976): 182-211. Reprinted 38.2, 3 (1999): 259-287 www.almaden.ibm.com/journal.
7. Gilb, Tom, and Dorothy Graham. *Software Inspection*. Addison-Wesley, 1993. Japanese Translation, Aug. 1999.
8. Mills, Harlan D. *IBM Systems Journal*. 1980. Also republished IBM Systems Journal, Nos. 2 and 3, 1999.
9. Cotton, Todd. "Evolutionary Fusion: A Customer-Oriented Incremental Life Cycle for Fusion." *Hewlett-Packard Journal* 47.4 (Aug. 1996): 25-38.
10. Shewhart, Deming, Juran 1920s.
11. Deming, W. Edwards. *Out of the Crisis*. Cambridge: MIT CAES Center for Advanced Engineering Study, 1986.
12. Florac, William A., Robert E. Park, and Anita D. Carleton. *Practical Software Measurement: Measuring for Process Management and Improvement*. Pittsburgh: Software Engineering Institute, 1997 www.sei.cmu.edu.
13. Akao, Yoji. *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Cambridge: Productivity Press, 1990.
14. Gilb, Tom. *Principles of Software Engineering Management*. Boston: Addison-Wesley, 1988.
15. Gilb, Tom. *Competitive Engineering*. ~~Addison-Wesley: United Kingdom, 2000~~ ~~www.resultplanning.com~~.
16. Cusumano, Michael A., and Richard W. Selby. *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*. The Free Press (a division of Simon and Schuster), 1995.
17. Woodward, Stuart. "Evolutionary Project Management." *IEEE Computer* Oct. 1999: 49-57.
18. Kaplan, Craig, Ralph Clark, and Victor Tang. *Secrets of Software Quality, 40 Innovations From IBM*. McGraw Hill, 1944.
19. Crosby, Philip B. *Quality Is Still Free: Making Quality Certain in Uncertain Times*. McGraw Hill, 1996.

About the Author

Tom Gilb has been a freelance consultant since 1960 and is the author of nine books, including "Software Metrics," "Principles of Software Engineering Management," "Software Inspection," and the forthcoming "Competitive Engineering." Gilb teaches and consults worldwide with major multinational clients including Nokia, Ericsson, Motorola, HP, IBM, BAE Systems, Philips, Sony, Canon, Intel, and Microsoft and does pro bono training and

consulting for the Department of Defense, United Kingdom, NATO, and the Norwegian Defense.

~~Iver Holtersvei 2~~
NO-1410
Kolbotn, Norway
Phone: +47 66 80 46 88
E-mail: tom@gilb.com
