No Cure No Pay:

How to Contract for Software Services

Tom Gilb
RPL
Iver Holtersvei 2, NO-1410 Kolbotn, Norway

Abstract. 50% of all software projects are total failures and another 40% are partial failures according to widely quoted surveys in UK, USA and Norway. Large government projects in all 3 countries have been reported with spectacular failure and expense to taxpayers (Royal Academy of Engineering and British Computer Society 2004). What is the problem? Most discussions have centered on improving the software engineering process itself: better estimation, better requirements, better reuse and better testing. No doubt all those can be improved. However, I suggest the motivation to improve them needs to be put in place first. Think about it. Most of these failures have been fully paid for! We not only pay well for failure, but the bigger the failure, the more people get paid!

My suggestion is simple. Pay only when defined results are provably delivered. This requires several things:

- Contracts that release payment only for meaningful results;
- The ability to define those results, particularly qualitative ones, and particularly the organizational ones;
- The ability to deliver those results incrementally, thus proving capability at early stages and continuously.

Note: This paper specifically addresses the software problem, but I am sure that the ideas here apply to the wider systems engineering problem to some interesting degree as well.

Defining the !Cure"

We write contracts, and we write requirements for projects, but these are often useless for the following reasons:

- We define the wrong things;
    - We *define* valueless things! Such as the following:
        - 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.

Of course all this is *management* work (not software engineering), and management consistently fails to manage properly. They make consistently bad assumptions that the software or IT system will deliver benefits. But they do not control the delivery of those benefits. Why is management so bad at this? Because they have not been trained to do anything other than this,, at any level of management training. Top managers are thus mostly at fault for not ensuring that ‘Pay for Results’ is the policy, and that managers are trained to ensure that money is not wasted on failed IT projects.

We define too-large pieces of work: we define work delivery packages in terms of years. We should define them instead in terms of quarters, months and weeks. We would then see early if any planed value can be delivered to real stakeholders, and if not, we would have chance to correct the situation or remove the incompetent supplier.

We choose contractors too early, based on the wrong criteria. We choose based on size and ‘reputation’ (not the reputation for delivering successfully!). We should instead choose suppliers based in proven ability in the past, and on our project to deliver. We could, as I suggested recently (2004) 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)

The request for proposal will sound like this:

“We invite you to tender for a contract to <build software/deliver an IT system>. The contract will be based on a ‘Value Payment’ system. This means that we will define what *we* expect in terms of testable and measurable values from the system. We will pay only when that value is satisfactorily and provably delivered. We will not pay for effort put in, and we will not

pay for sub-specification results. If you are focused on delivering us the results we agree on, then you can earn money independently of the costs to you. Efficient suppliers can earn more than usual. Inefficient suppliers would not. 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 template in the appendix. It is a framework for subcontracting as the 'Evolutionary Value Delivery' step level.

The essential ideas in a 'No Cure, No Pay' Contract are:

- Payment is totally dependent on proven delivery of our Value Definition;
- Estimates for delivering the value will be made by the Contractor in advance;
- We will accept some level of cost overrun compared to the estimates, when actual costs exceed the estimate (for example, 100%). Above that, the Contractor pays such excess costs;
- 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).

Motivating your Team to Contract for Results

So, how do we motivate the cultural change to contracting for results? The Policy examples in the Appendix give some motivating points you can use. Here are some supplementary ideas:

- It is our professional/ethical/legal duty towards shareholders/customers/taxpayers to make absolutely sure we do not waste money on projects that do not deliver expected results;
- The Supplier/Contractor has the technical/managerial/economic/experiential competence to control the results/cost ratio for us. That is why we are contracting with them. If they cannot contract for results, we should not do business with them;
- It is our professional obligation to always clearly/testably/measurably define what we expect to get for our money, in advance. Anything less is incompetence and unprofessional. If we cannot do that, we should not have the job of commissioning any use of resources for our company;
- We do not intend to *ever* be a party to failure. If our contracting process fails to deliver, it is *our* fault, *not* the supplier's fault. Any failure will be *structurally* (step size one week, for example) kept to a minimum, and failure will result in reconsidering the supplier's continued participation;
- We need to prioritize high value results early one, and lock them in. We cannot get

involved in primarily and initially building superstructures that fail;

- Maybe we, the Customer, should make this even clearer by bonus or rewards for employees and teams that manage value deliveries successfully.

You are going to have to *train* your *own* staff to do this:

- Give them a clear policy;
- Give them training in quantification of values delivered;
- Give them training is testing and measurement of value delivered;
- Give them training in requirements specification;
- Give them useful templates (see appendix);
- Give them access to experts in house or externally in doing this.

Motivating Suppliers to Contract For results

We can expect suppliers to initially resist such contracts, for all the obvious reasons:

- They are unaccustomed to this way of contracting;
- They will probably require permission to do so from their own top management;
- They are not sure what they will be getting into. They have no such experience;
- They like the old idea of getting paid millions, even if the result is useless for the customer.

So, we are going to have to motivate them!

- Refuse to do business on any other terms;
- Make sure they know you are offering this business to their competitors, and that their competitors have indicated willingness to do it;
- Employ multiple competing suppliers at the same time. Let them know that the biggest share of the business will go to the best value provider;
- Get your top management to make it clear to their top management, that this is the new way of doing business;
- Get some early examples of supplier success, using this method, to tell hesitant suppliers about;
- Reduce and eliminate business flow from the suppliers who do not actually deliver results, and redirect the flow to those who do.

Quantifying Qualitative Results

We all know how to specify results about storage capacity, transaction throughput, and response time. The difficulty for most Customers is specification of *quality* ideas – popularly called ‘ilities’ for obvious reasons.

This is not difficult, *if* you have some training, and some quantification templates. We have

found that absolutely all qualitative aspects of software can be expressed quantitatively in a practical way. We have also seen that, once you find a quantification, there is always a practical way to test the level of that quality in practice. *One customer of mine reported that they needed between 30 minutes and 2 hours to do the tests [Johansen05].*

Few professionals, managers or engineers, have any training in quantification of qualities. So, this is one of the real barriers to paying only for results: the quantification of the *qualitative* values.

The process of quantification is simple in principle, and is mostly willpower and common sense.

If it varies, it can be quantified, by definition.

Many qualities are 'complex'; meaning they actually involve a set of sub-quantities. You might have to list the sub-qualities, and quantify them individually.

Maintainability

- Problem Recognition Time
- Administrative Delay Time
- Tool Collection Time
- Problem Analysis Time
- Correction Hypothesis Time
- And many more! (Fix, Check, Tests)

Write the word 'Scale:' and define the quality variation in such as way that you can put a number on it.

Intuitiveness: Type: Quality Requirement.

Scale: % of tasks where defined Users need no help from people, manuals, help lines etc. to correctly and immediately carry out defined Tasks.

Then put two numbers on it

Past: 30%.

Goal [Next Version] 80%.

(this is from a real example by one of our customers (Johansen 2005) – they reached their goal in 3 months.

The value of reaching such a technical goal, as the 80% intuitiveness, can be understood, perhaps even roughly estimated by

- Estimating the time saving for an average task (say 3 minutes)
- Estimating the amount of such savings for a year: say 3 x 100 users X 10 times/day X 200 days = 600,000 minutes or 10,000 hours/year.

If you are stuck for ideas, Google searches will usually give you 20,000 to 60,000 hits!

If you would like the basics see my Scales chapter in Competitive Engineering (Gilb 2005)

Table 1: A real example of weekly control of quantified quality results. From FIRM [Johansen05] in the 9th week, 95% of the planned improvement is delivered (2x planned result) using a design idea called !Recoding". Similar results were achieved every week for the first 24 weeks of using this new method of project control.

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

Direct and Indirect Results

There is a critical distinction between the performance characteristics of a software or IT system itself (System Performance) – and the higher level organizational impacts that system performance characteristics are expected to deliver.

For example: a system might be designed to a Usability level so good that it takes 10% of older systems to learn and to use. This system performance characteristic has an *organizational* value in relation to

- How many people it affects (Savings Population)
- How often they use it (Savings Frequency)
- The time period over which the value is derived (Lifetime Value)

As a practical matter we will probably pay the supplier for the System Performance. It is then our problem to exploit that performance, and pull in the real savings.

An option is, however, to contract for the *actual savings through time*, as the product is used. This could be for a limited time period (60 days, a year) as a kind of field trail acceptance test. It could conceivably be a lifetime of system Royalty to be paid to the supplier. This is not unlike the periodic software-use fees, or licenses, in widespread practice, except, they would instead be based on *value delivered*, rather than *product used*.

Table 2: This is a conceptual example. Three goals (performance targets) and two resource targets are having the real impacts on them tracked, as steps are delivered. The same IE table is also being used to specify the impact estimates for the future planned steps. So at each step, the project can learn from the reality of the step's deviation from its estimates. Plans and estimates can then be adjusted and improved from an early stage of the project.

Step Target Requirement	**Step 1** Plan % (of Target)	Actual %	Deviation %	**Step 2** to **Step 20** Plan %	Plan % cumulated to here	**Step 21 [CA, NV, WA]** Plan %	Plan % cumulated to here	**Step 22** [all others] Plan %	Plan % cumulated to here
Performance 1	5	3	-2	40	43	40	83	-20	63
Performance 2	10	12	+2	50	62	30	92	60	152
Performance 3	20	13	-7	20	33	20	53	30	83
Cost A	1	3	+2	25	28	10	38	20	58
Cost B	4	6	+2	38	44	0	44	5	49

Decomposing Projects into Small Result Deliverables

One useful way to make sure that value is *really* delivered is to make it happen early and often. In my book, that translates to next week and every week. Many of my customers from Hewlett Packard to the 70 person software product developer 'Future Information Management Research' do exactly that.

Nobody is actually against early and frequent delivery of stakeholder value. The problem is that people have not been trained, or led, to decompose their long term visions of improvement, into a series of smaller incremental tasks.

We are not talking about 'building' things incrementally. That is unavoidable. We are talking about actually delivering value to real people, time savings, money savings, making life better for them in their work or hobbies. We are talking about the very thing that the failed IT projects forgot to do. Give Results!.

Technical 'experts' will come up with 100 excuses as to why things need to take a whole year (actually 3 years before they admit failure). They cannot conceive of next week, and every week. It is outside their culture and their training, and their management. So, they need

encouragement, and training!

Management has to declare that hereafter things will be done in early, short, frequent increments. Those who can – will get work. Those who cannot – are not useful – valuable.

Then the technological planners need to be taught the art of decomposing their big visions into incremental value deliveries.

I have observed about 20 principles (Gilb05, Ch. 10 Evo, also in Appendix here) for decomposition. The 'big trick' is to focus on one value aspect at a time, and ask the simple question, "What can we do, in a week or so, to actually move measurably forward towards our Goal?"

Strangely there always seems to be an answer in practice! If you don't get one, you either need to study decomposition methods more deeply, or listen to those who can find solutions better.

But if a certain step takes longer, don't panic. Maybe do something useful to deliver other results in parallel, or just be patient for a week or two! What is unacceptable is to plan for a few years and delay a few years and fail. Anything is better than that old habit of IT people.

From our main point of view, weekly or short increments are merely a useful discipline that makes us really focus on delivering value. The main point is that we focus on really satisfying stakeholders, and on keeping our promises to them.

A Basic Evolutionary Planning Policy

1. Financial Control: No project cycle can exceed 2% of total initial financial budget before delivering some measurable, required results to stakeholders.

2. Deadline Control: No project cycle can exceed 2% of total project time (For example, one week for a one year project) before delivering some measurable, required results to stakeholders.

3. Value Control: The next step should always be the one that delivers best stakeholder value for its costs

Figure 1. Policy Statement: For No Cure No Pay Project Management

Summary

One way to avoid software project failure is to refuse to pay for failure. This will motivate software suppliers to make use of already well-known and well-practiced methods for successful IT and software projects (Larman 2003, Gilb 2005).

There are two key ideas that too many people do not practice (due to lack of training and/or poor management). The first is the quantification of the values expected by stakeholders of the system, especially the 'qualitative' aspects. This gives the basis for payment.

The second idea is to divide all large projects into an incremental series of smaller projects. This means roughly weekly or 2% of current large projects, per step of value delivery. Each increment must attempt to increase some aspect of stakeholder value, in the direction of the longer-term requirements. This small step discipline makes sure that suppliers really know what they are doing, and are really focused on value delivery, rather than their traditional concern for technical construction.

Top management must lead this culture change: the software technologists have consistently failed for decades, and the problem has never been technological.

References

Gilb, Tom, *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*, Elsevier Butterworth-Heinemann, 2005. ISBN 0750665076.

Gilb, Tom and Johansen, Trond, "From Waterfall to Evolutionary Development (Evo): How we rapidly created faster, more user-friendly, and more productive software products for a competitive multi-national market", INCOSE Proceedings, July 2005. Also published in EuroSPI Proceedings, November 2004.

Larman, Craig and Basili, Victor R., "Iterative and Incremental Development: A Brief History", IEEE Computer, June 2003. Pages 2-11.

Royal Academy of Engineering and British Computer Society, The Challenges of Complex IT Projects, April 2004. ISBN 1-903496-15-2. URL: `www.raeng.org.uk`.

Biography

Tom has been an independent consultant, teacher and author since 1960. He works mainly with multinational clients; helping improve their organizations, and their systems engineering methods.

Tom's latest book is 'Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage' (Summer 2005).

Other books are (with Dorothy Graham) 'Software Inspection' (1993) and 'Principles of Software Engineering Management' (1988). His 'Software Metrics' book (1976, Out of Print) has been cited as the initial foundation of what is now CMMI Level 4.

Tom's key interests include business metrics, evolutionary delivery, and further development of his planning language, 'Planguage'. He is a member of INCOSE and is an active member of

the Norwegian chapter NORSEC. He participates in the INCOSE Requirements Working Group, and the Risk Management Group.

Email: Tom@Gilb.com

URL: http://www.Gilb.com

End of formal paper, the rest is a practical appendix that can be dropped if page limitations dictate.

Appendix: can be deleted if paper length considerations dictate.
(but it is a shame to delete such practical material for the reader!)

Sample Contract

Drop In 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 (Larman 2003).

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.

Planning Templates

Evo Step a Result Delivery Step: Here is a template made for this client to document each Evo step:

Evolutionary Delivery Step Plan (the Form)

Buyer Requirements

Functional Requirements

Benefit/Quality/Performance Requirements

Reference Tag:_____

Ambition Level: _____

Quantification Scale:_____

Meter [END STEP ACCEPTANCE TEST] ___

Past Level [<when?, where?>] ___. Source: _____.

Fail Level [[<when?, where?>] _____. Source:____

Goal level [[<when?, where?>_____. Source: _____

Cross Reference to more detail:

REPEAT THE ABOVE TEMPLATE FOR ALL DELIVERABLE RESULTS

Resource Constraints:

Calendar Time:

Work-Hours:

Qualified People:

Money (Specific Cost Constraints for this step):

Other Constraints

Design Constraints

Legal Constraints

Generic Cost Constraints

Quality Constraints

Assumptions:

Dependencies:

The Resource Constraints can be done for the sum of each defined result, or for each one of them.

Design:
Technical Design (for Benefit Cost requirements)

Reference Tag:
Description (or pointer to tags defining it):
Expected impacts: <specify level of impact, on tagged results>
Evidence (for expected level of impacts)
Source (of evidence)

Reference Tag:
Description (or pointer to tags defining it):
Expected impacts: <specify level of impact, on tagged results>

Evidence (for expected level of impacts)
Source (of evidence)

Reference Tag:
Description (or pointer to tags defining it):
Expected impacts: <specify level of impact, on tagged results>
Evidence (for expected level of impacts)
Source (of evidence)

Reference Tag:
Description (or pointer to tags defining it):
Expected impacts: <specify level of impact, on tagged results>
Evidence (for expected level of impacts)
Source (of evidence)

Test Design
Supplier Test Plan:
Customer Acceptance Testing Plan:
First day trial:
30 Day trial:

Documentation Design:
Training Design:

Estimates:
Estimated Cost $
Estimated work-hours

Actual Cost $
Actual Work-hours

Reasons for differences:

Cost
Work-hours
Signoffs
Customer accepts and supports the plan (esp. requirements)
Customer Accepts that requirements are met during first trial day (Invoice can be sent): _____signature
Comments:
Changes desired (new requirements):
Customer accepts that Invoice can be paid for this increment : _____ sign.

Policy Examples

The Customer's Project Policy

Version 0.2
Owner: The Supplier Project Leader for The Customer
Author: Tom@Gilb.com ©

Objective: to create a relationship for The Customer which

• removes problems caused by dynamically changing and evolving requirements.
• gives The Customer rapid actual usable system improvement.
• gives The Customer complete control of cost (no cure no pay).
• gives The Customer complete flexibility to change requirements to suit current insights into their critical needs.
• gives The Supplier the ability to focus on delivering satisfactory real improvements to the way The Customer does business.
• creates a sound basis for a happy long term relationship between the parties based on delivered value for money, as judged by The Customer.

- THE EVOLUTIONARY RESULT DELIVERY POLICY

1. The current project will continue by planning to deliver customer usable/evaluatable system improvements in approximately weekly intervals.
2. The precise increment requirements will be settled at the week beginning from a menu of interesting options, as selected by the Customer.
3. The increment will be intentionally scaled down to probably be doable within the scope of a week, but shorter or longer cycles may be agreed as needed.

4. The agreed incremental result delivery will be normally delivered to the client for their appraisal and use by Friday morning.
5. The Customer will preliminarily evaluate it by end of day.
6. If it meets agreed requirements the customer will formally indicate that an invoice for the incremental effort can be sent, payable within 30 days. If not accepted, reasons will be given in writing, which relate to failure to meet agreed written specifications.
7. Payment is effectively due when no hidden problems are discovered in the next 30 days in which payment is due, which invalidate acceptance. I.e. that it did not in fact meet specified requirements. Written notice giving details of failure to meet specified requirements will be given as a basis for holding up payment.
8. The Supplier is responsible for rectifying any previously unacceptable delivery increments before proceeding to do any later work on the project.

Agile Project Control of Value delivery. Source: Gilb05.

- **A Simplified Evo Process**
  - *Background: A simplified version of the Evo process to use on small projects. It also serves to help understand the larger, full-scale Evo process.*
  - Tag: Simplified Evo. Version: October 7, 2004. Owner: TG. Status: Draft.
  -
  - **Process Description**
    - Gather from all the key stakeholders the top few (5 to 20) most critical goals that the project needs to deliver. Give each goal a reference name (a tag).
    - For each goal, define a scale of measure and a 'final' goal level. For example: *Reliable: Scale: Mean Time Before Failure, Goal: >1 month.*
    - Define approximately 4 budgets for your most limited resources (for example, time, people, money, and equipment).
    - Write up these plans for the goals and budgets (*Try to ensure this is kept to only one page*).
    - Negotiate with the key stakeholders to formally agree the goals and budgets.
    - Plan to deliver some benefit (that is, progress towards the goals) in *weekly* (or shorter) increments (Evo steps).

        - Implement the project in Evo steps. Report to project sponsors after each Evo step (weekly, or shorter) with your best available estimates or measures, for each performance goal and each resource budget. O*n a single page,* summarize the *progress to date* towards achieving the goals and the costs incurred.
- **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.

Decomposition Principles

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

1. Believe there is a way to do it, you just have not found it yet![1]
2. Identify obstacles, but don't use them as excuses: use your imagination to get rid of them!
3. Focus on some usefulness for the stakeholders: users, salesperson, installer, testers or customer. However small the positive contribution, something is better than nothing.
4. Do <u>not</u> focus on the design ideas themselves, they are distracting, especially for small initial cycles. Sometimes you have to ignore them entirely in the short term!
5. Think one stakeholder. Think 'tomorrow' or 'next week.' Think of one interesting improvement.
6. Focus on the results (You should have them defined in your targets. Focus on moving *towards* the goal and budget 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.

---

[1] Working within many varied technical cultures, I have never found an exception to this since 1960 – there is always a way!

9. Don't be afraid that the stakeholders won't like it. If you are focusing on the results[2] they want, then by definition, they should like it. If you are not, then do!
10. Don't get so worried about "what might happen afterwards" that you can make no practical progress.
11. You cannot foresee everything. Don't even think about it!
12. If you focus on helping your stakeholder in practice, now, where they really need it, you will be forgiven a lot of 'sins'!
13. You can understand things much better, by getting some practical experience (and removing some of your fears).
14. Do early cycles, on *willing local mature* parts of your user/stakeholder community.
15. When some cycles, like a purchase-order cycle, take a long time, initiate them early (in the 'Backroom'), 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', stakeholders, or end user. They probably have dozens of suggestions.
19. Talk with end users and other stakeholders in any case, they have insights you need.
20. Don't be afraid to use the old system and the old 'culture' as a launching platform for the radical new system. There is a lot of merit in this, and many people overlook it.

-

12 Tough questions: (Source Gilb05)

Twelve Tough Questions

A way to sum up this paper for managers!

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, w*hy* 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?

A full paper on this is available at www.Gilb.com
•

Version: 13 April 2006 TG