Project Failure Prevention: 10 Principles for Project Control

Tom Gilb Tom@Gilb.com

Copyright © 2005 by Tom Gilb. Published and used by INCOSE with permission.

Abstract: It is now well-known and well-documented that far too many projects fail totally or partially, both in engineering generally (Morris 1998) and software engineering (Neill and Laplante 2003). I think everybody has some opinions about this. I do too, and in this paper I offer some of my opinions, and I hope to lend some originality to the discussion. As an international consultant for decades, involved in a wide range of projects, and involved in saving many 'almost failed' projects, my basic premises in this paper are as follows:

- We specify our requirements unclearly;
- We do not focus enough on ensuring that the system design meets the requirements.

INTRODUCTION

The principles for project control can be summarized by a set of ten principles, as follows:

P1: CRITICAL MEASURES: The critical few product objectives (performance requirements) of the project need to be stated measurably.

P2: PAY FOR RESULTS: The project team must be rewarded to the degree they achieve the critical product objectives.

P3: ARCHITECTURE FOR QUALITY: There must be a top-level *architecture* process that focuses on finding and specifying appropriate design strategies for enabling the critical product objectives (that is, the performance requirements' levels) to be met on time.

P4: CLEAR SPECIFICATIONS: Project specifications should *not* be polluted with dozens of defects per page; there needs to be specification quality control (SQC) with an exit condition set that there should be less than one remaining major defect per page.

P5: DESIGN MUST MEET THE BUSINESS NEEDS: Design review must be based on a 'clean' specification, and should be focused on whether the designs meet the business needs.

P6: VALIDATE STRATEGIES EARLY: The high-risk strategies need to be validated early, or swapped with better ones.

P7: RESOURCES FOR DESIGNS: Adequate resources need to be allocated to deliver the design strategies.

P8: EARLY VALUE DELIVERY: The stakeholder value should be delivered early and continuously. Then, if you run out of resource unexpectedly, proven value should already have been delivered.

P9: AVOID UNNECESSARY DESIGN CONSTRAINTS: The requirements should not include unnecessary constraints that might impact on the delivery of performance and consequent value.

P10: VALUE BEFORE BUREAUCRACY: The project should be free to give priority to value delivery, and not be constrained by well-intended processes and standards.

PRINCIPLES

P1: CRITICAL MEASURES: The critical few product objectives (performance requirements) of the project need to be stated measurably.

The major reason for project investment is always to reach certain levels of product performance. ‘Performance’ as used here, defines how good the system function is. It includes:

- *Qualities* - how well the system performs;
- *Resource Savings* - how cost-effective the system is compared to alternatives such as competitors or older systems;
- *Workload Capacity* - how much work the system can do.

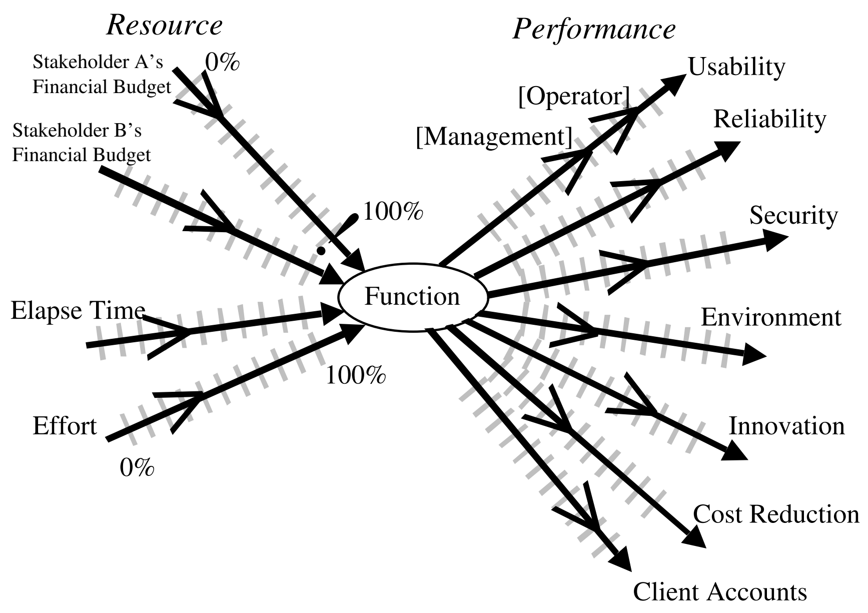


Figure 1. The ‘product’ of a project will want to attain a number of critical performance requirements. Serious project control necessitates clear agreement about the set of performance levels. The project can then focus on delivering these levels, within the available resources

In practice, you need to be concerned with the 5 to 20 'most-critical' product performance requirements (For example, see Figure 1). These are the critical dimensions that determine if a project has been a success or failure, in terms of the *product* produced by that project. I choose to make a clear distinction between the *project* characteristics (like team spirit and budget overrun) and the project *product* characteristics, and to focus here on the *product* characteristics as the decisive success or failure concepts. I am not concerned with 'the operation was a success, but the patient died' view of systems engineering.

I observe, in project after project, that I almost never see what I would call a well-written set of top-level requirements. The problems, which I perceive, include:

- The critical product characteristics are often not clearly identified *at all;*
- They are often identified only in terms of some *proposed design* (like 'graceful file degradation' to quote a recent one) to achieve requirements (rather than 'file availability', a requirement area);
- They are often pitched at an inappropriately *technical* level ('modularity' rather than 'flexibility');
- When they *are* identified they are often specified in terms of *'nice words'* (for example, 'state-of-the-art security') rather than a *quantified* engineering specification (such as '99.98% reliability');
- Even when some quantification is given - it often lacks *sufficient detail and variety* to give engineering control. For instance including the *short*-term goals – not just the final goals, and including the different goals for the a variety stakeholders – not just the implied system user. I usually see no explicit statement of the rationale for the performance levels specified.

If the critical success factors for the projects output are not well specified, then it does not matter how good any consequent process of design, quality control, or project management is. They cannot succeed in helping us meet our primary product requirements. See Figure 2 for an example of a quantitative specification of a performance requirement. This is the level of detail that I consider appropriate.

Requirement Tag: Interoperability:

Interoperability: defined as: The ability of two or more IS, or the subcomponents of such systems, to exchange information and services, and to make intelligent use of the information that has been exchanged < JSP.

Vision: The system shall make business application data visible across the boundaries of component sub-systems <- SRS 2.2.7.

Source: SRS Product ID [S.01.18, 2.2.7].

Version: October 2, 2001 11:29.

Owner: Mo Cooper.

Ambition: Radically much better Interoperability than previous systems.

Scale: Seconds from initiation of a defined [Communication] until fully successful intended intelligent [Use] is made of it, under defined field [Conditions] using defined [Mode].

Meter [Acceptance] <A realistic range of at least 100 different types of Communication and 100 Use and 10 Conditions> <- TG.

=== Benchmarks ============== *Past Levels* ================================

Past [UNNICOM, 2001, Communication = Email From Formation to Unit, Use = Exercise Instructions, Conditions = Communication Links at Survival]: <infinite> seconds <- M Cxx.

Conditions: defined as: Field conditions, which might threaten successful use.

Record [DoD, 1980?, Communication = Email From Formation to Unit, Use = Exercise Instructions, Conditions = Communication Links at Survival]: 5 seconds <- ??

Trend [MoD UK, IT systems in General, 2005, Mode = {Man transporting paper copy on motorbike, or any other non-electronic mode}]: 1 hour?? <- TG.

=== Targets ================== *Required Future Levels* ============================

Goal [DoD, 2002, Communication = Email From Formation to Unit, Use = Exercise Instructions, Conditions = Communication Links at Survival]: 10 seconds?? <- ??

Figure 2. Specifying a performance requirement using Planguage. This example is a first draft from a real project

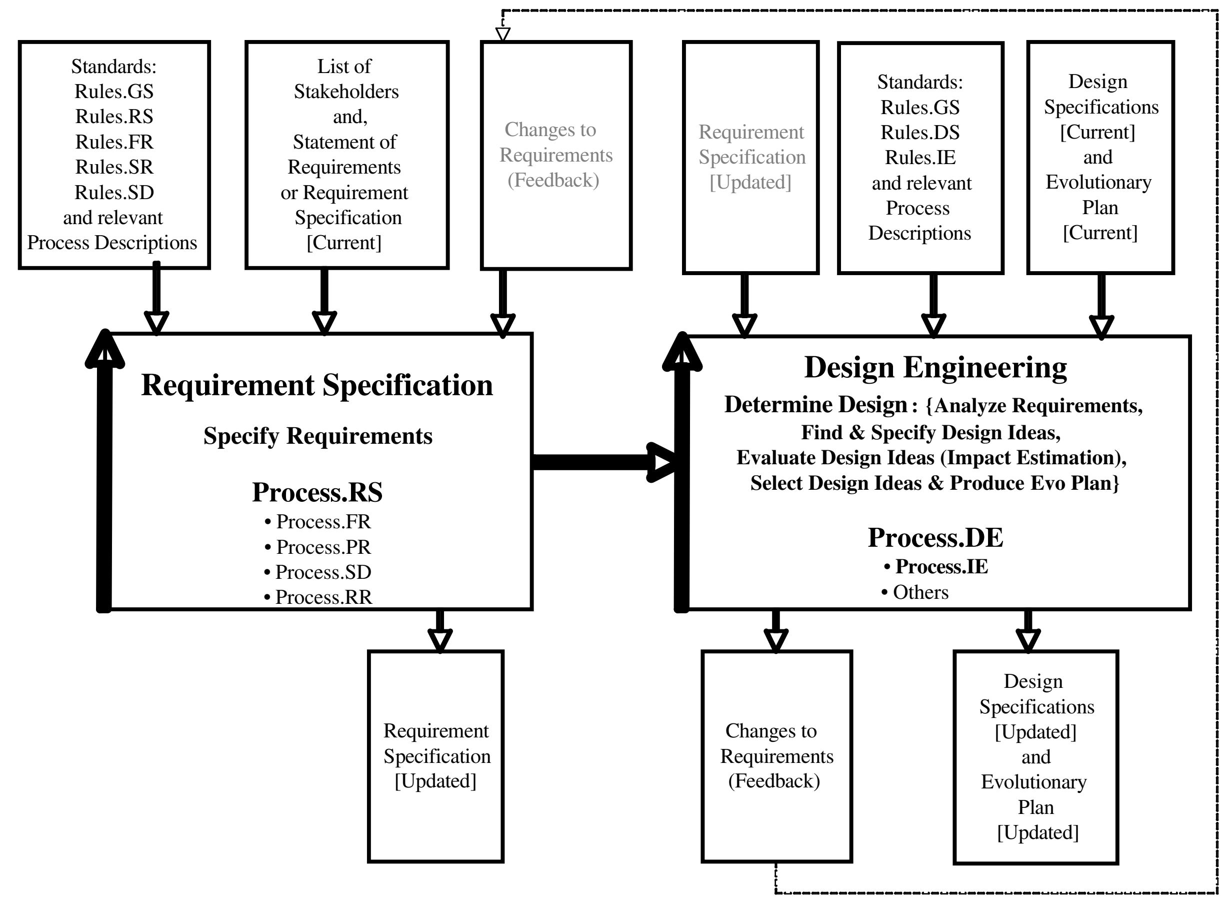
P2: PAY FOR RESULTS: The project team must be rewarded to the degree they achieve the critical product objectives.

What do we do for the project team when they *fail* to deliver the specified requirements, or when they use more time and money than budgeted? We *reward* them by continuing to pay them.

Now, if being out of control on performance and costs was entirely beyond their powers, then 'payment for effort' might be a reasonable way to do things. But I believe that we would be much better off if there were clear rewards for reaching targeted performance levels within budgeted resources. And lack of reward, for failing.

We would, of course, have to make it a 'fair' game. The project team would have to voluntarily agree that the performance goals were in fact realistic, in relation to the resources they have, or can get, to deliver them. We would also need to resist the temptation to dictate work processes or to specify designs, which in the view of the project team, would stop them from achieving the project goals.

Of course if we still cannot specify the performance goals quantitatively, then no amount of motivation and freedom will get a project team to move in the right direction. They don't even know what that direction is.



Standards: Rules.GS Rules.RS Rules.FR Rules.SR Rules.SD and relevant Process Descriptions
List of Stakeholders and, Statement of Requirements or Requirement Specification [Current]
Changes to Requirements (Feedback)
Requirement Specification [Updated]
Standards: Rules.GS Rules.DS Rules.IE and relevant Process Descriptions
Design Specifications [Current] and Evolutionary Plan [Current]
Requirement Specification
Specify Requirements
Process.RS
• Process.FR
• Process.PR
• Process.SD
• Process.RR
Design Engineering
Determine Design: {Analyze Requirements, Find & Specify Design Ideas, Evaluate Design Ideas (Impact Estimation), Select Design Ideas & Produce Evo Plan}
Process.DE
• Process.IE
• Others
Requirement Specification [Updated]
Changes to Requirements (Feedback)
Design Specifications [Updated] and Evolutionary Plan [Updated]

Figure 3. The design engineering process depends on the quality of the requirement specification. The standard for the requirement specification can be set partly by the specification rules, and partly by the defect density tolerance (that is, how seriously the rules are taken as measured by a quality control process). The standard processes and tags (for example, 'Rules.GS') refer to the

standards published in the CE manuscript (Gilb 2005)

P3: ARCHITECTURE FOR QUALITY: There must be a top-level *architecture* process that focuses on finding and specifying appropriate design strategies for enabling the critical product objectives (that is, the performance requirements' levels) to be met on time.

If you do not have a clear quantified set of top-level critical requirements, then an architecture process is bound to fail. The architect cannot compare *their* design idea's expected performance and cost attributes with the clear performance and cost *requirements*, they need to have for clear judgments about design ideas.

Even if the requirements are perfectly quantified and clear, that is not sufficient. The architecture designs themselves must be specified in sufficient detail to enable an judgment that they will probably provide the necessary levels of performance and cost impacts (See Figure 3).

Tag: OPP Integration.

Type: Design Idea [Architectural].

============ Basic Information ==

Version:

Status:

Quality Level:

Owner:

Expert:

Authority:

Source: System Specification [Volume 1 Version 1.1, SIG, February 4. – Precise reference <to be supplied by Andy>].

Gist: The X-999 would integrate both 'Push Server' and 'Push Client' roles of the Object Push Profile (OPP).

Description: Defined X-999 software acts in accordance with the <specification> defined for both the Push Server and Push Client roles of the Object Push Profile (OPP).
Only when official certification is actually and correctly granted; has the {developer or supplier or any real integrator, whoever it really is doing the integration} completed their task correctly.
This includes correct proven interface to any other related modules specified in the specification.

Stakeholders: Phonebook, Scheduler, Testers, <Product Architect>, Product Planner, Software Engineers, User Interface Designer, Project Team Leader, Company engineers, Developers from other Company product departments, which we interface with, the supplier of the TTT, CC. "Other than Owner and Expert. The people we are writing this particular requirement for"

============ Design Relationships ==

Reuse of Other Design:

Reuse of this Design:

Design Constraints:

Sub-Designs:

============== Impacts Relationships ==

Impacts [Intended]: Interoperability.

Impacts [Side Effects]:

Impacts [Costs]:

Impacts [Other Designs]:

Interoperability: Defined As: Certified that this device can exchange information with any other device produced by this project.

============= Impact Estimation/Feedback ==

Impact Percentage [Interoperability, Estimate]: <100% of Interoperability objective with other devices that support OPP on time is estimated to be the result>.

============== Priority and Risk Management ====================================

Value:

Figure 4. An example of a real draft design specification that attempts to both have necessary detail, and to make some assertions and estimates about the effects of the design on requirements. Much more could be specified later.

P4: CLEAR SPECIFICATIONS: Project specifications should not be polluted with the usual dozens of defects per page; there needs to be specification quality control (SQC) with an exit condition set that there should be less than one remaining major defect per page.

I regularly find that any requirement specification given to me by a new customer, even if it is approved and being used, has between 80 and 180 ‘major’ defects. This is normally a ‘shock’ for the people involved. How can there be so many? We measure them by asking colleagues of the specification author, and often the author too, to check a sample ‘logical’ page (that is, 300 non-commentary words) of a requirements specification, and count any violations of these simple rules:

- Clear enough to test;
- Unambiguous to the intended readership;
- No unintentional design in the requirements.

I then ask participants to evaluate if each rule violation (defect) is serious enough to potentially cause delays, costs, and product defects. They are asked to classify any that are serious, as ‘major’ defects. The range of majors found per sample page, in about 15 minutes of checking, is usually from 3 to 23 per person. I find that small groups of 3 to 4 people typically find about double the most defects found by a single individual. For example, if the greatest number of defects found by one person is 15, then a small group would have about 30 unique majors to report. But these 30 are only one third of what is actually in the spec right now. The checking process is about 30% effective. Consequently there are something like 90 major defects present in the page, of which the small group can find only a third in about 15 minutes.

Most people immediately agree that this is far too many. It is. And it is unnecessary! How polluted a requirements specification would you think is acceptable? If you are professional you will finally set a limit of *no more than one remaining major defect per page* before you release the specification for use in design or test planning.

Total Defects	**Majors**	**Design (part of Total and M+m)**
M+m	**M**	**Design**
41	24	D=1
33	15	D=5
44	30	D=10
24	3	D=5

Table 1: An example from a 30 minute checking of real project requirements at a Jet Engine Company by 4 managers. The sample page was called 'Non-Functional requirements. By extrapolation the team found about 60 of a total in the page of about 180 major defects. The 'M+m' is majors and minors. The 'D' numbers are the number of designs in the requirements

My experience (since 1988 working with aircraft drawings in California) is that if you set such exit conditions (max. 1 major/page) and take them seriously, then individuals will learn to reduce their personal injection of major defects by about 50% for each SQC cycle they go through. Within about 7 cycles, the individual engineer will be able to specify such a clean specification that it will exit first time. Some people can achieve this with fewer cycles.

The conclusion is that because we do not carry out even simple inexpensive sampling measures of specification pollution, and we do not set a limit to the pollution, we live in a world of engineering that is normally highly polluted. We pay for this by project delays, and poor product quality.

P5: DESIGN MUST MEET BUSINESS NEEDS: Design Review must be based on a 'clean' specification, and should be focused on whether the designs meet the business needs.

Before we can review a design specification regarding relevance, we must review it for ‘compliance to rules of specification’, that assures us about the *intelligibility* of the specification.

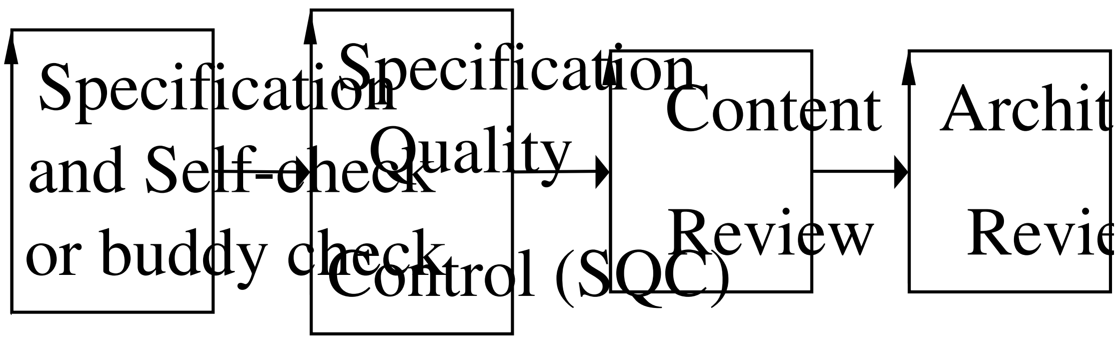

Specification and Self-check or buddy check
Specification Quality Control (SQC)
Content Review
Archit
Revie

Figure 5. Four stages of checking a design specification

For example, there is no point in reviewing a design specification (such as ’Reusable Modules’), when

- The specification is unnecessarily ambiguous;
- There is no assertion of which requirements the design is supporting;
- There is no assertion or estimation as to how well the design is supporting those requirements;
- There are no resource estimates (costs) for the design.

To put it more directly: Check that a design specification is *well written* first, only then do you have a basis for checking to see if it is a *good* design. I believe it is unreasonable to judge a design idea itself on the basis of a poorly written specification. Only well-specified designs should be evaluated by senior responsible people. Badly specified designs – defined as those not following our own design specification rules – need to be returned to the designer for rework. See Figure 6 for an example of real design rules designed to force us to specify designs in sufficient detail. When these rules are followed, then, I believe we have the basis for deciding whether a design is appropriate in relation to its requirements.

Rules: Design Specification

Tag: Rules.DS. Version: October 7, 2004. Owner: TG. Status: Draft.

Note: Design specifications are either for optional design ideas (possible solutions) or required design constraints (that is, actual requirements AND consequently, pre-selected solutions).

Base: The rules for generic specification, Rules.GS apply. If the design idea is a design constraint (a requirement), the rules for requirement specification, Rules.RS also apply.

R1: **Design Separation**: Only design ideas that are intentionally ‘constraints’ (*Type: Design*

Constraint) are specified in the *requirements*. Any other design ideas are specified separately (*Type: Design Idea*). Note all the design ideas specified as requirements should be explicitly identified as 'Design Constraints.'

R2: **Detail**: A design specification should be specified in *enough detail* so that we know precisely what is expected, and do not, and cannot, inadvertently assume or include design elements, which are not actually intended. It should be 'foolproof'. For complex designs, the detailed definition of its sub-designs can satisfy this need for clarity, the high level design description does not need to hold all the detail.

R3: **Explode**: Any design idea, whose impact on attributes can be better controlled by detailing it, should be broken down into a list of the tag names of its elementary and/or complex sub-design ideas. Use the parameter 'Definition' for Sub-Designs. If you know it can be decomposed; but don't want to decompose it just now, at least explicitly indicate the potential of such a breakdown. Use a Comment or Note parameter.

R4: **Dependencies**: Any known dependencies for successful implementation of a design idea need to be specified explicitly. Nothing should be assumed to be 'obvious'. Use the parameter, Dependency (or Depends On), or other suitable notation such as [qualifiers].

R5: **Impacts**: For each design idea, specify *at least one* main performance attribute impacted by it. Use an impact arrow '->' or the Impacts parameter. Comment: At early stages of design specification, you are just establishing that the design idea has some relevance to meeting your requirements. Later, an IE table can be used to establish the performance to cost ratio and/or the value to cost ratio of each design idea.

Example:

Design Idea 1 -> Availability.

Design Tag 2: Design Idea.

Impacts: Performance X.

R6: **Side Effects**: Document in the design specification any side effects of the design idea (on defined requirements or other specified potential design ideas) that you expect or fear. Do this using explicit parameters, such as Risks, Impacts [Side Effect] and Assumptions.

Examples:

Design Idea 5: Have a <circus> -> Cost A.

Risk [Design Idea 5]: This might cost us more than justified.

Design Idea 6: Hold the conference in Acapulco.

Risk: Students might not be able to afford attendance at such a place?

Design Idea 7: Use Widget Model 2.3.

Assumption: Cost of purchasing quantities of 100 or more is 40% less due to discount.

Impacts [Side Effects]: {Reliability, Usability}.

Do not assume others will know, suspect or bother to deal with risks, side effects and assumptions. Do it yourself. Understanding potential side effects is a sign of your system engineering competence and maturity. Don't be shy!

R7: **Background Information**: Capture the Background information for any estimated or actual *impact* of a design idea on a performance/cost attribute. The evidence supporting the impact, the level of uncertainty (the error margins), the level of credibility of any information and, the source(s) for all this information should be given as far as possible. For example, state a previous project's experience of using the design idea. Use Evidence, Uncertainty, Credibility, and Source parameters. Note the source icon (<-) usually represents the Source parameter. Comment: This helps 'ground' opinions on how the design ideas contribute to meeting the requirements. It is also preparation for filling out an IE table.

Example:

Design Tag 2 -> Performance X <- Source Y.

R8: **IE Table**: The set of design ideas specified to meet a set of requirements should be validated at an early stage by using an Impact Estimation (IE) table.

Does the selected set of design ideas produce a good enough set of expected attributes, with respect to all requirements and any other proposed design ideas? Use an IE table as a working tool when specifying design ideas and also, when performing quality control or design reviews on design idea specifications.

R9: **Constraints**: No single design specification, or set of design specifications cumulatively, can violate any specified constraint. If there is *any* risk that this might occur the system engineer will give a suitable warning signal. Use the Risk or Issues parameters, for example.

R10: **Rejected Designs**: A design idea may be declared 'rejected' for any number of reasons. It should be retained in the design documentation or database, with information showing that it was rejected, and also, why it was rejected and by whom.

Example:

Design Idea D: Design Idea.

Status: Rejected.

Rationale [Status]: Exceeds Operational Costs.

Authority: Mary Fine. Date [Status]: April 20, This Year.

Figure 6. Specification rules for design, which lay a proper basis for judging the power and cost of the design in relation to the requirements. See also Chapter 7 in (Gilb 2005)

P6: VALIDATE STRATEGIES EARLY: The high-risk strategies need to be validated early, or swapped with better ones.

It should be possible to identify your high-risk strategies (Note, 'strategies' are also known as designs, solutions and architectures). They are the ones that you are not sure of the impact levels on performance and cost. They are the ones that can potentially ruin your project, if they turn out to be worse than you are expecting. If you try to estimate all impacts of a given strategy on an Impact Estimation table, and get estimates such as 50%±40% (100% = Goal level attained), then you have exposed a high-risk design. If the credibility estimate (a defined Impact Estimation method) on a scale of 0.0 to 1.0 is low, like under 0.5, then you also have a high-risk strategy. If no one will guarantee the result in a binding contract, then you also have a high-risk strategy.

Engineers have always had a number of techniques for validating risky strategies early. Trials, pilots, experiments, and prototypes are common tactics. One approach I particularly favor is scheduling the delivery of the risky strategy in an early evolutionary delivery step, to measure what happens – and thus get rid of some of the risk. Early evolutionary delivery steps usually integrate a new strategy with a real system, and with real users, and are therefore more trustworthy, than, for example, an expert review panel, which is relatively theoretical.

Policies for Risk Management

Explicit Risk Specification: All managers/planners/engineers/testers/quality assurance people shall specify any uncertainty, and any special conditions, which can imaginably lead to a risk of deviation from defined target levels of system performance. This must be done at the earliest opportunity in writing, and integrated into the main plan.

Numeric Expectation Specification: The expected levels of all quality and cost attributes of the system shall be specified in a numeric way, using defined scales of measure, and at least an outline of one or more appropriate 'meters' (that is, a proposed test or measuring instrument for determining where we are on a scale).

Conditions Specified: The requirements levels shall be qualified with regard to when, where and under which conditions the targets apply, so there is no risk of us inadvertently applying them inappropriately.

Complete Requirement Specification: A *complete* set of all critical performance and cost aspects shall be specified, avoiding the risk of failing to consider a single critical attribute.

Complete Design Specification and Impact Estimation: A complete set of designs or strategies for meeting the complete set of performance and cost targets will be specified. They will be validated against all specified performance and cost targets (using Impact Estimation Tables). They will meet a reasonable level of safety margin. They will then be evolutionarily validated in practice before major investment is made. The Evo steps will be made at a rate of maximum 2% of total project budget, and 2% of total project timescales, per increment (Evo step) of designs or strategies.

Specification Quality Control, Numerically Exited: All requirements, design, impact estimation and evolutionary project plans, as well as all other related critical documents, such as contracts, management plans, contract modifications, marketing plans, shall be 'quality controlled' using the Inspection method (Gilb 1993). A normal process exit level shall be *that 'no more than 0.2 maximum probable major defects per page can be calculated to remain, as a function of those found and fixed before release, when checking is done properly'* (that is, at optimum checking rates of 1 logical page or less per hour).

Evolutionary Proof of Concept Priorities: The Evolutionary Project Management method will be used to sense and control risk in mid-project. Dominant features will include:

• 2% (of budget and time-to-deadline) steps;

• High value-to-cost steps, with regard to risk, prioritized asap;

• High risk strategies tested 'offline to customer delivery', in the 'backroom' of

the development process, or at cost-to-vendor, or with 'research funds' as opposed to using the official project budget.

Figure 7. Policy Ideas for Risk Management

P7: RESOURCES FOR DESIGNS: Adequate resources need to be allocated to deliver the design strategies.

It is not sufficient that your design strategies will meet your *performance* targets. We have to have the *resources* needed to implement them. And the resources used must not result in an unprofitable situation, even if we can get them. The resources we must consider are both for development and operation, even decommissioning. The resources are of many types, and include money, human effort and calendar time.

I observe that it is unusual for me to see design specifications with detailed cost calculations at the level of *individual designs*. We do not even seriously try to consider individual design idea costs (at best we estimate a total cost); so it is not surprising that we constantly exceed the resource constraints that apply to our projects.

P8: EARLY VALUE DELIVERY: The stakeholder value should be delivered early and continuously. Then, if you run out of resource unexpectedly, proven value should already have been delivered.

Projects may fail because they run out of time or money and have delivered nothing. They can then only offer hope of something, at the cost of additional money and time – and note that this addition is typically estimated by the people who have already demonstrated they do not know what they are promising.

Designs->	**Contract**	**Supplier**	**Motive**	**Architect**	**Parts Used**	*Sum %Impacts*
Requirements						
Performance						
Quality 1	0%	100%	50%	30%	-20%	160%
Quality 2	100%	50%	0%	20%	50%	220%
Costs						
Investment Cost	5%	10%	1%	10%	110%	136%
Operational Cost	5%	50%	20%	1%	10%	86%
Staff Resource	10%	20%	10%	5%	0%	45%
Performance to Cost Ratio	100/20	150/80	50/31	50/16	30/120	

Table 2: An Impact Estimation table structure simplified example that helps us consider cost elements, while looking at the performance impacts. Consequently

we can see the performance to cost ratio. The % estimates refer to % of meeting a performance level target, or to % of a budgeted cost

The smart management solution to this common problem is to demand that projects are done evolutionarily. That means there will be consciously-planned early (first month) and frequent (perhaps weekly, or 2% of budget) attempts to deliver measurable value to real stakeholders.

Most people have never been subject to this discipline, and they have not learned the theory of how to do it in practice. But there are decades of practical proof in the software and systems engineering world that this works. (Larman and Basili 2003, Larman 2003).

P9: AVOID UNNECESSARY DESIGN CONSTRAINTS: The requirements should not include unnecessary constraints that might impact the delivery of performance and consequent value.

It is all too common for projects to focus on a particular technical solution or architecture, and not to focus on the actual end results they expect to get from the technical 'design'. They end up locking themselves into the technical solution – and rarely get the results they expected. Remember the high failure rate of projects?

The primary notion in planning any project, and in contracting suppliers to deliver all or part of it, is to focus entirely on the top few critical results. The critical results have to be quantified within the requirements and made the subject of contract conditions (such as, 'No cure, No pay').

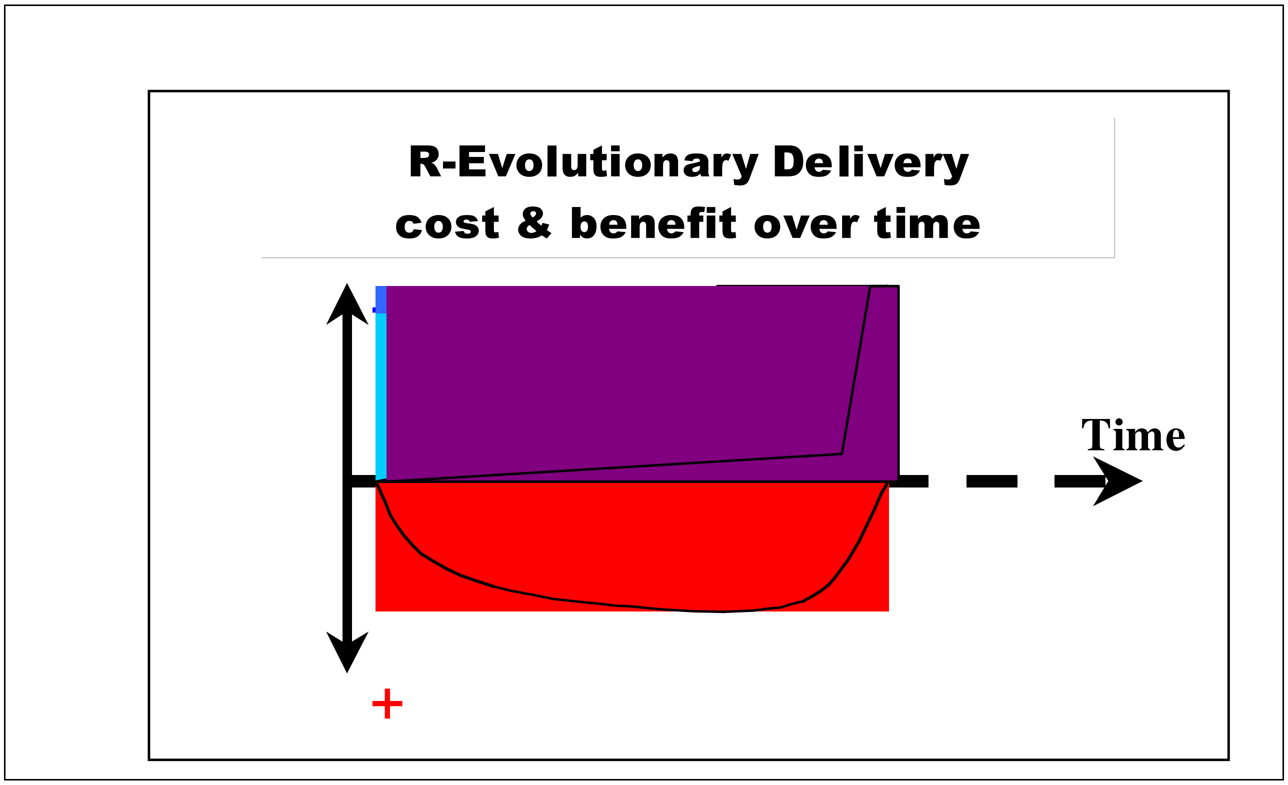


Figure 8. Conceptual view of delivery of stakeholder benefits early and cumulatively. From Kai Gilb's book Manuscript (Gilb, K. 2005)

Technology Policy

- Use the technology which best satisfies your requirements.
- If you are responsible for results, you have total discretion about the means.
- Those who dictate constraints are responsible for the constraint effects.
- Make sure the risk of technology you choose is understood, controlled and profitable.
- Satisfy stakeholder priorities in profitable sequence, within available resources.

Figure 9. A technology policy the author suggested to one of his clients, a major electronics telecom organization

P10: VALUE BEFORE BUREAUCRACY: The project should be free to give priority to value delivery, and not be constrained by well-intended processes and standards.

There was a time when software and IT were 'Wild West'. Anybody who could program, did things as they knew best. Sometimes, we are not far in many places from that model today. However in other places, the need to get higher consistent standards of professionalism, has swung the pendulum too far the other way. Processes and standards like the Software Engineering Institute Capability Maturity Model Integration (CMMI 2002) are thorough and well intended. But almost no such recommended frameworks and processes encourage or permit focus on the *main results* of a project. Consequently there is a great, inevitable, danger that this *results focus* will be lost in practice. Everywhere I look, I see that result – no results focus – worldwide – with or without the distraction of CMMI and the like. This includes the Agile Methods (Larman 2003). My recommendation to attempt to refocus is outlined in Figure 10.

A Simple Evolutionary Project Management Method

Project Process Description

1. Gather from all the key stakeholders the top few (5 to 20) most critical performance (including qualities and savings) goals that the project needs to deliver. Give each goal a reference name (a tag).

2. For each goal, define a scale of measure and a 'final' goal level. For example: *Reliability: Scale: Mean Time Between Failure, Goal: >1 month.*

3. Define approximately 4 budgets for your most limited resources (for example, time, people, money, and equipment).

4. Write up these plans for the goals and budgets (*Try to ensure this is kept to only one page*).

5. Negotiate with the key stakeholders to formally agree the goals and budgets.

6. Plan to deliver some benefit (that is, progress towards the goals) in *weekly* (or shorter) increments (Evo steps).

7. Implement the project in Evo steps. Report to project sponsors after each Evo step (weekly, or shorter) with 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.

- Based on numeric feedback, and stakeholder feedback; *change whatever needs to be changed to reach goals.*

8 When all goals are reached: "Claim success and move on" [Gerstner, 2002]. Free the remaining resources for more profitable ventures

Project Policy for Simple/Agile Evo Projects

1. 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.

2. The team will be paid and rewarded for 'benefits delivered' in relation to cost.

3. The team will find their own work process, and their own design.

4. 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. For more detail, see (Gilb 2003, Gilb 2004).

Figure 10.

REFERENCES

Bahill, A. Terry and Henderson, Steven J., "Requirements Development, Verification and Validation Exhibited in Famous Failures", *Systems Engineering*, Vol. 8, No. 1, 2005 pp. 1-14

CMMI, Capability Maturity Model Integration, Carnegie Mellon Software Engineering Institute, 2002, http://www.sei.cmu.edu/cmmi/ [Last Accessed April 2005].

Gilb, Kai, *Evo*, 2005. Draft manuscript at http://www.gilb.com

Gilb, Tom and Graham, Dorothy, *Software Inspection*, Addison Wesley, 1993.

Gilb, Tom, "Software Project Management: Adding Stakeholder Metrics to Agile Projects", *Novática*, Issue 164, July-August 2003. (Special Edition on Software Engineering - State of an Art, Guest edited by Luis Fernández-Sanz. Novática is the journal of the Spanish CEPIS society ATI (Asociación de Técnicos de Informática.) See http://www.upgrade-cepis.org/issues/2003/4/upgrade-vIV-4.html In Spanish: http://www.ati.es/novatica/2003/164/nv164sum.html

Gilb, Tom, "Adding Stakeholder Metrics to Agile Projects", *Cutter IT Journal: The Journal of Information Technology Management*, July 2004, Vol. 17, No.7, pp31-35. See http://www.cutter.com.

Gilb, Tom, *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering using Planguage*, Due June 2005, Elsevier Butterworth-Heinemann. ISBN 0750665076.

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

Larman, Craig, *Agile and Iterative Development: A Manager's Guide*, Addison Wesley, 2003. See Chapter 10 on Evo.

Morris, Peter W. G., *The Management of Projects*. London: Thomas Telford. 1998. ISBN 072771693X. 358 pages. USA: The American Society of Civil Engineers.

Neill, Colin J., and Laplante, Phillip A., "Requirements Engineering: The State of the Practice", *IEEE Software*, November/December 2003, pp. 40-45.

BIOGRAPHY

Tom Gilb is the author of 'Competitive Engineering: A Handbook for Systems & Software Engineering Management using Planguage' (due June 2005), 'Principles of Software Engineering Management' (1988) and 'Software Inspection' (1993). His book 'Software Metrics' (1976) coined the term and, was used as the basis for the Software Engineering Institute Capability Maturity Model Level Four (SEI CMM Level 4). His most recent interests are development of true software engineering and systems engineering methods.

Tom Gilb was born in Pasadena CA in 1940. He moved to England in 1956, and then two years later he joined IBM in Norway. Since 1963, he has been an independent consultant and author. He is a member of INCOSE.

Author Contact: Tom@Gilb.com

Edited by Lindsey Brodie, lindseybrodie@btopenworld.com, Middlesex University. London

Version April 24 2005