Design Evaluation: Estimating Multiple Critical Performance and Cost Impacts of Designs

Tom Gilb
Tom@Gilb.com

Abstract: How should we evaluate someone's design suggestion? Is gut feel and experience enough for most cases? Is anything more substantial and systematic possible? This paper outlines a process for design evaluation, which assesses the impacts of designs towards meeting *quantified* requirements. The design evaluation process is viewed as consisting of a series of design filters.

INTRODUCTION

This paper proposes that:

- Design evaluation is primarily a matter of understanding the 'performance and cost' impacts of the design(s) numerically, in relation to quantified performance and cost requirements;
- Design evaluation needs to go through several maturity stages, but remains fundamentally the same question: *what does the design contribute to meeting our requirements?*

Prior to design evaluation, the requirements must be specified. All the performance and cost requirements must be specified quantitatively. The requirements should be subject to specification quality control (SQC). An entry condition into the design evaluation process should be that the requirement specification has successfully exited quality control with an acceptable level of remaining defects per logical page (for example, less than one remaining defect per 300 words).

The design evaluation process consists of several maturity stages, which can be viewed as design filters. These maturity stages include:

- **Value-based selection**: Select the requirements with the highest stakeholder value;
- **Constraint-based elimination**: Delete designs that violate constraints;
- **Performance-based selection**: Pick the most effective remaining designs;
- **Resource-based optimization**: Select the effective designs that are most efficient – effect/cost;
- **Risk-based elimination**: Evaluate designs based on performance and cost risks.

Each of these evaluation filters requires specification and estimation tools that are common sense, but are not commonly taught or employed. These tools are based on the defined Planning Language ('Planguage'), developed by the author for practical industrial use through many years (Gilb 2005a).

Let's now go through the design evaluation process in greater detail. I'll summarize the main points as a set of principles.

CLEAR SPECIFICATION OF REQUIREMENTS

Principle 1: A design can *only* be evaluated with respect to *specific* clear requirements.

The foundation of design evaluation is a set of clear requirements. Any evaluation of any design to try to ensure meeting vague requirements is going to be imprecise[1]. If you look around you, both inside the systems engineering community and outside it, you will observe that people commonly evaluate designs 'in general terms' (for example, "*that* design is the *most* user-friendly…"), rather than with respect to specific *immediate*, and *residual*[2] project or product requirements.

One of several reasons for this generalizing about 'good designs,' is that we are very vague with requirements specification. Our most critical requirements are typically unclear, and not quantified. In my consulting and teaching practice I see this happening worldwide. When I evaluate such requirements using SQC, there are invariably approximately 100 defects present per page, unless special effort has been made to eliminate them (Gilb 2005b). To the degree that is the case, we cannot readily expect anyone to perform a logical evaluation of the suitability of a given design for a fuzzy requirement. Consider the following questions:

> *How good is a 'Mac interface' for getting 'higher usability'? And how good is 'MAC OS X' compared to 'Windows' for 'higher security'?*

These are silly questions because the requirements are not clearly defined (Note also that the other security designs that will co-exist are not listed and analyzed). Now, requirements are not the primary subject of *this* paper[3]. So we need to be brief on them, so as to concentrate on design itself. But here are some of the things I would insist are necessary pre-requisites for being able to evaluate a design:

- All performance (including all qualities) and cost requirements are expressed quantitatively (with defined scales of measure). Not just nice sounding words.

Scale: Minutes for defined [Tasks] done by defined [People].

- All performance requirements must include at least one *target* (A target is a level we aim for) and at least one *constraint* (A constraint is a level we aim to avoid with our design).

Goal: 3 minutes. **Fail**: 10 minutes.

- All performance requirements must include explicit and detailed information regarding the *short-term* and the *long-term* timescales of expectation.

Goal [Release 1.0]: 3 minutes. **Fail** [Release 1.0]: 10 minutes.

- *A*ll relevant *constraints* on solving the design problem are specified complete, officially, explicitly, unambiguously, and clearly. This includes all notions of restrictions such as legal, policy, and cultural constraints. It also includes any known design constraints (such as from our own architecture specification). Constraints will consider all necessary aspects of development, operations, and decommissioning resources.

---

[1] This is not the same as demanding that the requirements are known upfront: requirements should not be 'frozen' and they should be allowed to evolve over time. The issue here is that the known or predicted requirements are expressed clearly.

[2] Residual requirements: Residual: Concept *359: The remaining distance to a target level from a benchmark or current level (From Planguage Glossary in (Gilb 2005a)). The point being that design is a sequential process of evaluating necessary designs, to add onto the current set of designs – and the only designs necessary at any point in the process, are designs that will move us from the performance levels we estimate we have reached, with the current set of designs, towards our required Goal levels of performance. A good analogy is the 'next chess move'

[3] See Gilb 2005e for further discussion on requirements.

<Name tag of the performance requirement or cost requirement>:

Ambition: <Give overall real ambition level in 5-20 words>.

Version: <Each requirement specification should have a version, at least a date, yymmdd>.

Owner: <The person or instance allowed to make official changes to this requirement>.

Type: <Performance|Cost Requirement>.

Stakeholder: { , , }. "Who can influence your profit, success or failure?"

Scale: <Defined units of measure, with [parameters] if you like>.

Meter [<qualify which version and level>]: <Specify how you will measure>.

====Benchmarks ========== the Past ==========

Past [<time>, <place>, <event>]: <Actual or estimate of past level> <- <Source of past data>.

Record [<time>, <place>, <event>]: <Actual or estimate of record level>] <- <Source of record data>.

Trend [<time>, <place>, <event>]: <Prediction of level> <- <Source of prediction>.

==== Targets ========== the Future Needs ==========

Wish [<time>, <place>, <event>]: <- <Source of wish>.

For Performance Requirements Only – Use 'Goal'

Goal [<time>, <place>, <event>]: <Target level> <- <Source of goal>.

Value [<stakeholder>]: <Refer to what this impacts or how much it creates of value>.

For Cost Requirements Only – Use 'Budget'

Budget [<time>, <place>, <event>]: <Target level> <- <Source of budget>.

Stretch [<time>, <place>, <event>]: <Motivating target level> <- <Source of stretch>.

==== Constraints ==========

Fail [<time>, <place>, <event>]: <- <Source of fail>. "Failure Point"

Survival [<time>, <place>, <event>]: <- <Source of survival>. "Survival Point"

Figure 1 shows a Planguage requirement template with hints. This gives some idea of the basic parameters that should be used to describe a performance or cost requirement quantitatively (Gilb 2005a)

There clear and complete requirements are a set of basic entry conditions to any design or architecture process. Without it a design process is like a fighter plane with no known enemy, like a passenger ship at sea with no destination port identified, or like a great invention with no market.

Design evaluation is quite simply about deciding how well a design meets the total set of requirements.

Principle 2: All designs have performance and cost attributes, but not necessarily the ones *you* require.

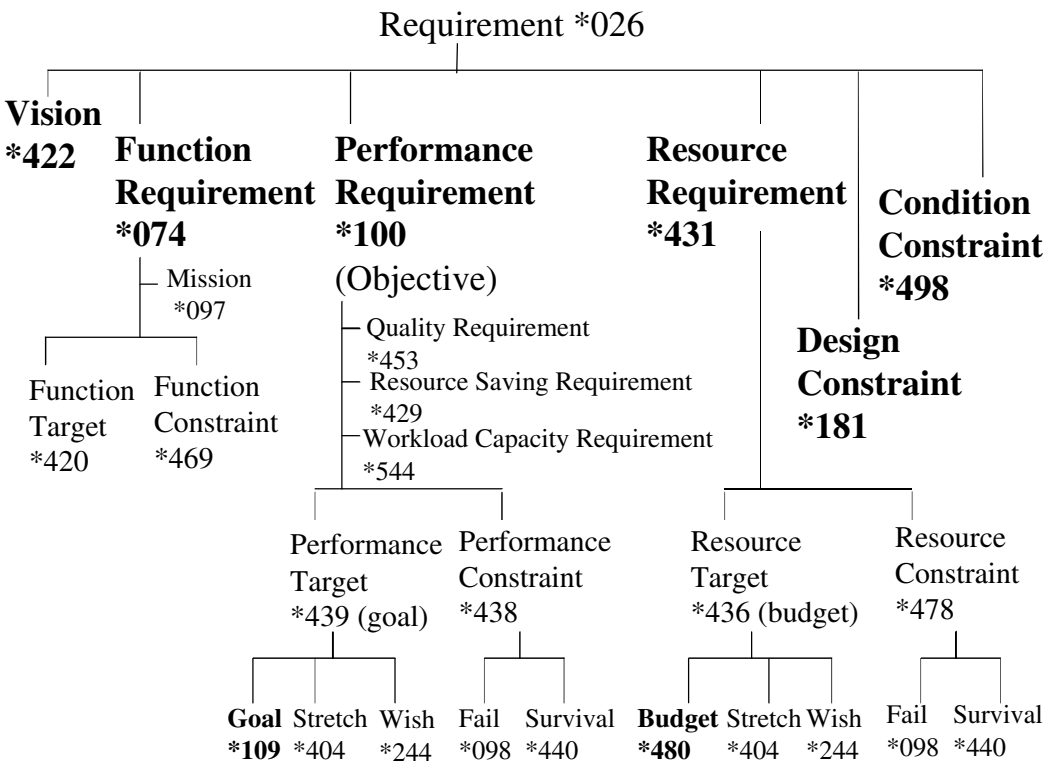

Figure 2 shows a map of the requirements concepts including a variety of constraint concepts. The *nnn are references to detailed definitions of these concepts in the Planguage Concept Glossary (Gilb 2005a)

VALUE-BASED SELECTION

Principle 3: The real value of a design to a stakeholder depends partly on the technical characteristics of the design, and partly on the planned, perceived and actual use of those characteristics in practice, over time.

The value of a design depends on the stakeholder view taken. The producer of a product has one view. The users of a product have another view. It is going to be the producer of a product who will directly and primarily evaluate designs from *their* point of view. They ideally will try to maximize their profitability or service delivery. The commercial producers will do this by maximizing the value delivered to their customers, so that their customers will 'return the favor' by paying well, in terms of price and volume. It should be possible to evaluate a design market, segment by market segment, for estimated sales or profit as a result of it. Ideally your marketing people would make such an evaluation. The *service* providers (such as military, space,

government) will worry about value to their stakeholders for money spent

If the marketing people are not involved or helpful, the technologist is left to look at the contribution of a given design to the *performance requirement* levels. Designs have value primarily as long as they help us move to the Goal levels, and perhaps to the degree they help us move to the Stretch levels (see Figure 1). Beyond those target levels, a design does not have any formally agreed value, because it is not formally required.

So we need to find the designs that satisfy the prioritized agreed target levels, at the lowest costs and risks available. I have developed an Impact Estimation (IE) method to help us see the contribution of design ideas to the requirement levels, the degree of risk involved and the corresponding development costs (See later, Table 1). So we can make a rational decision and present it to others.

CONSTRAINT-BASED ELIMINATION

Principle 4: It doesn't matter how good or how cheap a design is, if constraints forbid it.

We are assuming that there is a flow of one or more design ideas to be evaluated. The question of how we identify these design ideas is a separate topic. Before we go deeper into the design, we need to assess if any design idea is disqualified by any requirement.

We need to pass the design through the set of design filters known as constraints. The questions to be asked include:

- Does the design violate any specified design constraint?
- Does the design violate any condition constraint?
- Does the design violate any performance constraint or cost constraint?
- Does the design in *combination* with other design elements, adopted or projected, threaten to violate any constraint?

Because if a design violates, or threatens to violate, any defined constraint, a design needs to be set aside in favor of designs that do not. Later we could, if necessary, discuss relaxing a constraint, or risking or tolerating a constraint violation, in order to make use of an otherwise superior design, so we need to be careful about *permanently* discarding designs that initially violate some constraint. They might turn out to be the best design of all. So, 'set aside', preferably with *annotation about the constraint violation*.

Design X: <Detailed description>.
Status: Set Aside <- Tom, November 13, 2004.
Rationale: Threatens to violate cost constraints as it alone takes 90% of the budget.

I seriously suggest that all rejected designs be *formally kept* in the systems engineering documentation, with their status and rationale for rejection.

Principle 5: Designs should not be rejected permanently. The reasons for rejection should be clearly documented, the design specification kept; and the rejection possibly re-evaluated later.

PERFORMANCE-BASED SELECTION

For the set of proposed designs that survive constraint evaluation, the next step is to evaluate which ones have the *best set of impacts* on our *required performance target levels.*

Principle 6: The major capability of a design is its ability to contribute to required residual

performance levels.

We fail to evaluate designs in all critical dimensions. I find that systems and software engineers, in too many cases, do not even do a systematic evaluation of a design along a single performance dimension (such as 'Reliability'). But, even if they did do that, there is another evaluation problem to confront. Designs have potential impacts on many of our most critical performance requirement dimensions. Real systems seem to have about 20 to 40 performance dimensions that people are willing to set quantified requirements for, and to evaluate. One dimension is not enough. We need to look at:

- Other major secondary contributions to critical requirements;
- Possible negative side effects on the critical requirements.

We usually do not have good enough facts about the design impacts. Anything less than a thorough examination of the potential impacts of a design in *all* critical performance requirement

Strategy Comparison: Apples and Oranges

Objectives	Apples	Oranges	*Alternative Strategies*
Eater Acceptance From 50% to 80% of People	**70%**	**85%**	
Pesticide Measurement Reduce from 5% to1%	**50%**	**100%**	
Shelf-Life Increase from 1 week to 1 month	**70%**	**200%**	
Vitamin C Increase from 50mg to 100mg per day	**50%**	**80%**	
Carbohydrates Increase from 100mg to 200mg per day	**20%**	**5%**	
Sum of Performance	**260%**	**470%**	
Resources			
Relative Cost Local currency	**0.50**	**3.00**	
Sum of Costs	**0.50**	**3.00**	
Performance to Cost Ratio	**5.2**	**1.57**	

"Evidence" for these numbers should, of course, be available on a separate sheet (but not shown here)

Table 1: A symbolic example of evaluating two different 'designs' for 'which fruit to buy'). This is a simple Impact Estimation table application. The % estimated impact of a design is on a scale where 100% means the design brings us to the Goal level on time. 0% means there is no impact compared to some defined benchmark level, such as the previous system state.

dimensions, is irresponsible design engineering. The major initial outcome of any systematic quantitative evaluation in these many dimensions is, initially, 'shocking'. It turns out that even our most expert designers do not even *claim* to have any *factual* knowledge about *most* of the performance impacts of a design specification, in all our specified requirement dimensions! This may seem hopeless: 'Knowing that we do *not* know'. But in a sense it is the beginning of

wisdom, and there is a systematic approach to dealing with this ignorance – that is the subject of this paper. But we would do well to recognize this ignorance initially, clearly, and publicly, in our design engineering processes. Recognize the initial level of knowledge about a design, and then act cautiously as we progress the design towards serious commitment.

What is the alternative to a systematic initial design impact evaluation process? We do not have to 'act like engineers' and evaluate designs in a systematic and quantitative way. We can just 'decide to implement them and see what happens'. The problem there is that it may be too late to use better designs, and it would perhaps have paid off to do more engineering evaluation earlier.

There are interesting options between the extremes of 'full ignorance/high risk', and 'expecting perfect research data for all impacts of all design candidates'. For example evolutionary methods (Gilb 2005a, Larman 2003, Gilb 2005c, Johansen and Gilb 2005) may allow us to remove some of our ignorance about a design, at relatively low risk (By designing and implementing small Evo steps, we can ensure the maximum potential project loss 2% for a design that is a total failure). The fact that we rarely have the facts we need, in order to evaluate designs properly, is not a good reason to avoid trying to evaluate them quantitatively, before final commitment to using them. The lack of facts is a warning signal about risks. It can lead directly to more realistic expectations. It can also lead to risk mitigation tactics in contracting, alternative conservative design specifications, or lead to doing experimental steps to get needed data before scaling up – all traditional good engineering tactics.

Principle 7: A design will be best understood in terms of its multiple quantified impacts on your residual requirements.

How far should we go in evaluating a design? It is not enough, in my opinion to let your in-house expert loose, to make estimates of a design's impact on performance levels. They should be asked (in your systems engineering standards!) to document the basis for the estimates, and the basis for the uncertainty of their estimates. An example of doing this is given in Table 2 using the Impact Estimation method.

Notice that for each estimate we ask for the *uncertainty boundaries* (worst case/best case). We ask for *evidence* –the facts backing the estimate. We ask for the *source* of the evidence – a person or document for example. A reviewer of such estimates might be a skeptic, and want to check the evidence first hand. We can even rate the *quality of the basis for the estimate* using the 'credibility index' (say 0 for no credibility at all, and 1.0 for 100% credibility). Notice we can use the credibility-rating number to modify, by multiplication, the initial estimate, in the direction of a *more pessimistic* estimate. Better to be safe.

Principle 8: Designs must be evaluated with respect to uncertainty, and the level of risk you want to take.

I also like to get a simple estimate of the *cost* of the design, at least to become conscious of cost extremes.

	On-line Support	**On-line Help**	**Picture Handbook**	**On-line Help + Access Index**
Learning Past: 60minutes <-> Goal: 10minutes				
Scale Impact	5 min.	10 min.	30 min.	8 min.
Scale Uncertainty	±3min.	±5 min.	±10min.	±5 min.
Percentage Impact	110%	100%	60%	104%
Percentage Uncertainty	±6% (3 of 50 minutes)	±10%	±20%?	±10%
Evidence	Project Ajax: 7 minutes	Other Systems	Guess	*Other Systems* + Guess
Source	Ajax Report, p.6	World Report, p.17	John B	World Report, p.17 + John B
Credibility	0.7	0.8	0.2	0.6
Development Cost	120K	25K	10K	26K
Performance to Cost Ratio	110/120 = 0.92	100/25 = 4.0	60/10 = 6.0	104/26 = 4.0
Credibility-adjusted Performance to Cost Ratio (to 1 decimal place)	0.92*0.7 = 0.6	4.0*0.8 = 3.2	6.0*0.2 = 1.2	4.0*0.6 = 2.4
Notes: Time Period is two years.	Longer timescale to develop			

Table 2: A simplified example of using an impact estimation table to collect data about a single performance attribute. In this case, the performance attribute is 'Learning', which has a target level of 10 minutes. There are four design idea candidates (For example, 'On-line Support' is the tag of one design idea). We need to repeat this process for all other critical performance requirements. This is difficult because of lack of facts about most designs, in most dimensions. But the difficulty usefully makes us formally aware of design risks, and consequent project risks – which we can decide to mitigate by investigation, contracting, design or re-design

RESOURCE-BASED OPTIMIZATION

Of course, you do not understand a design idea, if you do not understand its costs. I mean the *entire range* of cost types (for example, effort, time, and money). I mean for the entire system lifespan.

Why do projects consistently run over time and budget, and you never seem to have enough people to do the job? (Gilb 2005d). One reason is that people fail to evaluate the costs of their designs. We do not practice 'design to cost'.

At least, if you have two or more promising design idea alternatives, you should consider

using the one with the least impact on your resource budgets.

Principle 9: Design ideas must also be evaluated with respect to the design costs' relation to our finite resources. Don't design what you can't afford.

But, I don't see people doing this in practice. I just see them running out of resources and instead of understanding that it might come from poor design practices, they blame other causes (such as too few resources).

RISK-BASED ELIMINATION

So, at this point, if you have followed the advice above, you might feel you have picked a winner set of design ideas with high performance impacts at low costs. But this is probably all based on *estimates*. Maybe those estimates are based on thin ice, such as rumor? Maybe experience data says the spread of *possible* actual design impacts on requirement levels is quite wide (like 10 minutes ± 9.9 minutes)? Maybe the 'technology behind the design' is not *that* new, but it has never been tried in *your* 'space vehicle', only in 'bicycles'? Enter the idea of 'risk evaluation'. What is the risk that your design idea, however hot it looks on paper, will not *really* work, or worse will ruin your entire project?

So, we need to ask the risk questions about each design idea. My favorite set of risk questions is my 'Twelve Tough Questions', given in Figure 3.

We have been asking some of these twelve analytical questions earlier in the design evaluation process above. But some are new. Who is responsible for making it work? Who is responsible if it does not work? Is their money where their mouth is?

I believe, in sharp contrast with the papers and textbooks that I have seen on risk management, that the risk analysis process is something that needs to be *intimately pervasive* in *every single* specification, in every detail of it. It must be part of what all systems engineers do every minute of their working life. Live it and breath it. Every systems engineering specification has an element of risk – or it would not be termed 'engineering' (Koen 2003).

We *need*, not to minimize risk, nor to reduce it to zero, but *to be constantly aware of risks.* We need to be constantly looking, waiting to pounce on risk if it shows signs of giving us trouble (Gilb 2003).

Principle 10: The evaluation of a design idea is a continuous process over a series of estimation and validation events. A lot of questions need asking, by a lot of people, and we need many good answers to evaluate a design.

Our systems engineering work should be totally robust so that no matter what happens we have a backup. We have a reasonable way out. We need to be so sensitive to the impacts of our designs that we know when we are threatened. We know early, because we worry early. We try things out early. We keep on measuring early as we make changes and add new designs cumulatively into the system.

We need above all not to trust a probability model of risk analysis. We need to take da Vinci's advice and try things out. See Figure 4. Much of his advice can be seen in the Evolutionary project management model, with its 2% increments, required for measurement, use of feedback, analysis of the feedback, and concept of changing the plan as necessary. We need to use evolutionary step planning to consciously sequence the riskiest elements for early integration and field trialing. Then if there is something wrong, we have lots of time to fix it.

Twelve Tough Questions

1. NUMBERS

   Why isn't the improvement quantified?

2. RISK

   What's the risk or uncertainty and why?

3. DOUBT

   Are you sure? If not, why not?

4. SOURCE

   Where did you get that information from? How can I check it out?

5. IMPACT

   How does your idea affect my goals?

6. ALL CRITICAL FACTORS

   Did we forget anything critical?

7. EVIDENCE

   How do you know it works that way?

8. ENOUGH

   Have we got a complete solution?

9. PROFITABILITY FIRST

   Are we going to do the profitable things first?

10. COMMITMENT

    Who's responsible?

11. PROOF

    How can we be sure the plan is working?

12. NO CURE

    Is it no cure, no pay?

Figure 3. A more detailed treatment of these questions is in a paper at http://www.gilb.com

Evolutionary project management (Evo) (Larman 2003, Larman and Basili 2003, Gilb 2005a, Gilb 2005c) is one of the greatest devices for risk management and for design evaluation with respect to risk, but Evo never, as far as I can see, made it into a paper or book on risk management, other than my own (Gilb 2003)! Evo allows you to evaluate one design at a time, and to evaluate them cumulatively, one at a time (Johansen and Gilb 2005).

In fact too many project management people have no clue what Evo really is. However, the US DoD finally understood it and adopted it (in 1995 with Mil Std 498 and on), calling it

'Evolutionary Acquisition'.

Principle 11: The best practical evaluation of design risks is by practical small step integration of the design, with measurement, feedback and analysis of its real performance and costs. Evolutionary evaluation helps us make better decisions about designs than any review committee will ever be able to make.

Curiosità:

Insatiably curious, unrelenting quest for continuous learning

Dimostrazione:

Commitment to test knowledge through experience, willingness to learn from mistakes.

Learning for ones self, through practical experience

Sensazione:

Continual refinement of senses. As means to enliven experience.

Sfumato:

Willingness to embrace ambiguity, paradox, uncertainty

Arte/Scienza:

Balance science/art, logic & imagination, whole brain thinking

Corporalità:

Cultivation of grace, ambidexterity, fitness, poise

Connessione:

Recognition & appreciation for interconnectedness of all things and phenomena.

Systems thinking

Figure 4. Da Vinci's Principles from *How to Think Like Leonardo da Vinci* by Michael Gelb. They describe the evolutionary principles for handling risk.

SUMMARY

Design evaluation needs a series of processes to determine the best-known design for a specific project. The foundation is a complete, clear and quantified set of requirements, against which to judge the design ideas. The second is a detailed design specification including justifications, assumptions, sources, and expected impacts. The third is the ability to see the expected effects of a set of design ideas, and their total impact on requirements. This initially can be achieved using an Impact Estimation (IE) table. However ultimately a design needs to be proven in practice by evolutionary implementation of the design ideas, while measuring their real cumulative impacts.

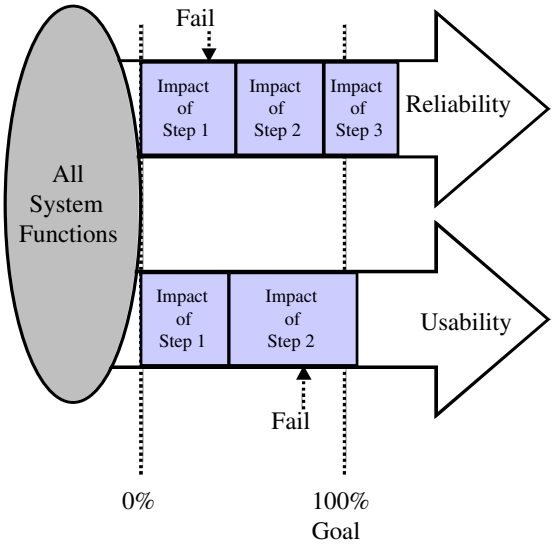

Figure 5. The step-by-step evolution of designs delivering impact to performance requirements

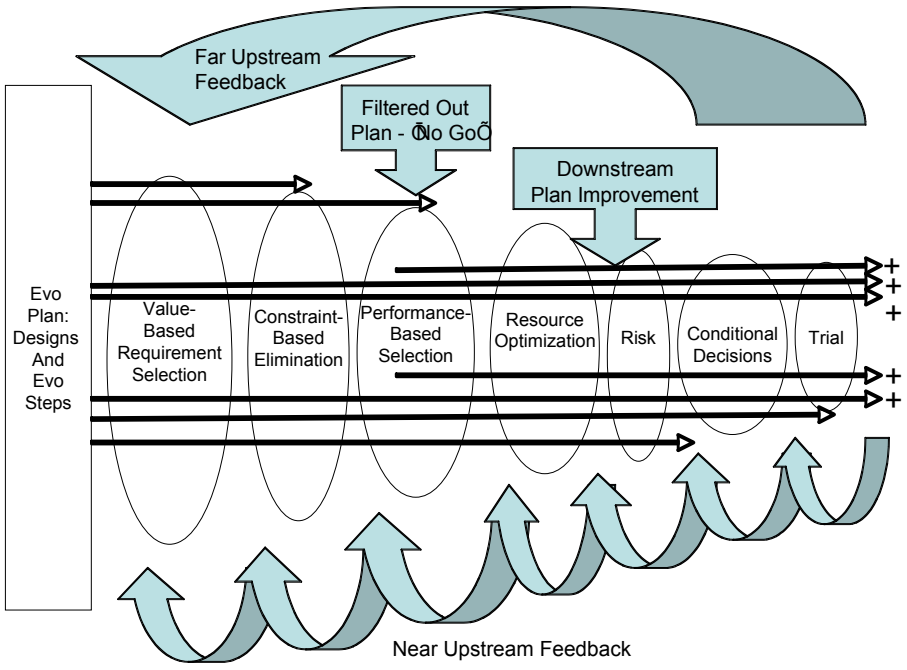

Figure 6. Relevance Control filters start after the QC filters of Rules and Exit make sure we have good presentation. The Relevance Control filters deal with questions of substance: how good is the plan in practice? The QC filters deal with the question, how well is the plan presented? The downstream plan improvements can come from any source, any reason, at any time, or any stage downstream (See also Gilb 2005a for details of SQC)

List of Principles

1: A design can only be evaluated with respect to specific clear requirements.

2: All designs have performance and cost attributes, but not necessarily the ones you require.

3: The real value of a design to a stakeholder depends partly on the technical characteristics of the design, and partly on the planned, perceived and actual use of those characteristics in practice, over time.

4: It doesn't matter how good or how cheap a design is, if constraints forbid it.

5: Designs should not be rejected permanently. The reasons for rejection should be clearly documented, the design specification kept; and the rejection possibly re-evaluated later.

6: The major capability of a design is its ability to contribute to required residual performance levels.

7: A design will be best understood in terms of its multiple quantified impacts on your residual requirements.

8: Designs must be evaluated with respect to uncertainty, and the level of risk you want to take.

9: Design ideas must also be evaluated with respect to the design costs' relation to our finite resources. Don't design what you can't afford.

10: The evaluation of a design idea is a continuous process over a series of estimation and validation events. A lot of questions need asking, by a lot of people, and we need many good answers to evaluate a design.

11: The best practical evaluation of design risks is by practical small step integration of the design, with measurement, feedback and analysis of its real performance and costs. Evolutionary evaluation helps us make better decisions about designs than any review committee will ever be able to make.

Figure 7 shows the list of principles presented in this paper

REFERENCES

Gilb, Tom, *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*, Elsevier Butterworth-Heinemann, Due June 2005a. ISBN 0750665076.
See also http://books.elsevier.com/companions

Gilb, Tom, "Agile Specification Quality Control: Shifting emphasis from cleanup to sampling defects." *Proceedings of INCOSE Conference*, Rochester NY USA, July 2005. Earlier version published as "Agile Specification Quality Control." *Cutter IT Journal*, Vol. 18 No. 1, page 35-39, January 2005b. See http://www.cutter.com [Last Accessed: April 2005].

Gilb, Tom, "Fundamental Principles of Evolutionary Project Management." *Proceedings of INCOSE Conference*, Rochester NY USA, July 2005c.

Gilb, Tom, "Project Failure Prevention: 10 Principles of Project Control." *Proceedings of INCOSE Conference*, Rochester NY USA, July 2005d.

Gilb, Tom, "Real Requirements: How to find out what the requirements really are." *Proceedings of INCOSE Conference*, Rochester NY USA, July 2005e.

Gilb, Tom, "Rule-Based Design Reviews." *Software Quality Professional*, Vol. 7, No. 1, 2004. pp. 4-13. See website for American Society for Quality: http://www.asq.org - member access only for recent papers.

Gilb, Tom, "Managing Your Project Risks in Requirements, Design and Development: Using the Planning Language." *Proceedings of INCOSE Conference*, Washington DC, 2003.

Gilb, Tom and Maier, Mark, "Managing Priorities: A Key to Systematic Decision Making." *Proceedings of INCOSE Conference*, Rochester NY USA, July 2005.

Goth, Greg, "New Air Traffic Control Software Takes an Incremental Approach." *IEEE Software*, July/August 2000, pp. 108-111.

Johansen, Trond and Gilb, Tom, "From Waterfall to Evolutionary Development (Evo) or How We Rapidly Created Faster, More User-Friendly, and More Productive Software Products for a Competitive Multi-national Market." July 2005. *Proceedings of INCOSE Conference*, Rochester NY USA, July 2005.

Koen, Billy Vaughn, *Discussion of The Method: Conducting the engineer's approach to problem solving*. Oxford University Press, January 2003. ISBN 0-195-15599-8. See also http://www.me.utexas.edu/~koen/ [Last Accessed: April 2005].

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.

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 1960, he has been an independent consultant and author. He is a member of INCOSE.

URL: www.Gilb.com

Author Contact: Tom@Gilb.com

The paper was edited by Lindsey Brodie, lindseybrodie@btopenworld.com

Version May 9 2005