Accurate estimation is impossible for complex technical projects, but keeping to agreed budgets and deadlines is achievable by using feedback and change. In other words, rather than trying to improve the initial project estimates, the budgets and deadlines must be set based on the value of delivery (not the cost), and then iterative re-engineering of product and process must be used to stay within acceptable resource bounds. Or, at least iteration must be used to get most of the expected value delivered, within the acceptable budgets and deadlines. This article explains the background to this approach and discusses its use, providing several examples.

Key words

estimation, evolutionary project management, Planguage

***SQP* References**

Impact of Defect Backlog on Product Release and Quality
vol. 10, issue 3
Hema Srikanth and Stephen H. Kan

Estimation: A Paradigm Shift Toward Dynamic Design-to-Cost and Radical Management

TOM GILB

INTRODUCTION

Accurate estimation of time and costs for complex systems and software projects is seemingly impossible to guarantee (Collins and Bicknell 1998; Craig and Brooks 2006; Grimstad, Jørgensen, and Moløkken-Østvold 2006). There are many unavoidable reasons for this. Even when estimation *seems* to work, this might just be a case of "stopping the effort" when the budget runs out. That method, however, is likely to result in delivering systems of unacceptably low quality. The main idea of this article is that there is a constructive alternative to such an unsatisfactory estimation process:

- Use process control (do a little, learn quickly, and change quickly) to rapidly and continuously tune anything and everything about the project, so prioritized resource budgets (such as time to market, money, and human resources) can be met.
- Consciously sacrifice "less-holy" things for "more-holy" objectives.

People are better off stipulating reasonable resource constraints (deadlines and cost budgets) and then learning to live within them. This is acceptable as long as one's highest performance and quality priorities are already satisfied, when resources run out. The rest of the requirements are, by definition, more marginal.

There are several reasons why it is difficult to estimate project costs accurately; it is because of the *inability* to:

- Define requirements (particularly multiple quality requirements) well enough, to estimate their costs, with useful accuracy.
- Specify the designs (also known as strategies or architecture) that are powerful enough to satisfy one's hopefully clear-and-complete requirements, well enough to know the design cost consequences.
- Collect, or have access to, experience data that would allow one to estimate costs, for well-specified designs, even without clear requirements.

Even with experience data, it would probably not help much, because the new project (typically, the critical cost-driving variables) would be different (from past experience) in some decisive way. In fact, time and cost estimates are not really necessary. Overall long-term estimates are an old custom, intended to prevent overruns and to give management some feeling that the job will get done in time, at a reasonable cost. Estimates do not, however, prevent overruns or assure value. What management needs is delivery of some critical system requirements in a *predictable* timeframe. They also need to be sure the project will be profitable and will not embarrass them with unexpected losses. This article describes an alternative way to achieve these management needs.

The Short Version: Constructive Suggestions for People Who Think They Want Estimates

Stipulate one or more *useful* deadlines from the management's point of view:

- Be specific about what has to be done by *each* deadline.
- Ask if these deadlines seem reasonable for the tasks prioritized.
- If necessary, make adjustments.

Then, stipulate the value of reaching each major ("Top 10") requirement. Make an outsourcing contract, and pay some percentage of that value only when that requirement is successfully delivered. Only do business with suppliers who consistently deliver value for money. Don't waste money on suppliers who make excuses, instead of value. "No cure, no pay" (Gilb 2006) is one way to motivate suppliers to provide value for money; otherwise, their motivation is to just keep billing (Craig and Brooks 2006).

For those who like short papers—that's it! Use the summary in Figure 1. However, some readers might like more explanation, more detail, more references, and some convincing arguments. If so, then read on.

FIGURE 1 Summary of the principles of resource control

The Risk Principles

1. **Drivers:** If one has not specified all critical performance and quality levels numerically, one cannot estimate project resources for those vague requirements.
2. **Experience:** If one does not have experience data about the resources needed for technical solutions, then one cannot estimate the project resources.
3. **Architecture:** If one implements the project solutions all at once, without learning their costs and interactions incrementally, one cannot expect to be able to understand the results of many interactions.
4. **Staff:** If a complex and large professional project staff is an unknown set of people, or changes mid-project, one cannot expect to estimate the costs for so many human variables.
5. **Sensitivity:** If even the slightest change is made, after an "accurate" estimation, to any of the requirements, designs, or constraints, then the estimate might need to be changed radically. And, one probably will not have the information necessary to do it, nor the insight needed to do it.

The Control Principles

6. **Learn small:** Carry out projects in small increments of delivering requirements, so one can measure results and costs against (short-term) estimates.
7. **Learn root:** If incremental costs for a given requirement level (and its designs) deviate negatively from estimates—analyze the root cause, and change anything about the next increments that might get things back on track.
8. **Prioritize critical:** Prioritize the most critical requirements and constraints: There is no guarantee one can achieve them all. Deliver "high value for resources-used" first.
9. **Risk fast:** Implement the design ideas with the highest value, with regard to cost and risk, early.
10. **Apply now:** Learn early, learn often, learn well, and apply the learning to the current project.

WHY ONE CANNOT EXPECT ESTIMATES TO BE CORRECT, ACCURATE, RELIABLE, OR TRUE

1. Drivers: If one has not specified all critical performance and quality levels numerically, one cannot estimate project resources for those vague requirements.

Costs are a result of the designs deployed to meet certain requirements. In particular, the cost-driving designs relate directly to performance and quality requirements. Costs are not quite so sensitive to functions or size.

It has long been understood, for example, COCOMO (Boehm et al. 2000), that there are a large number (dozens) of software cost drivers, such as availability. In one case, the 5ESS project (Lucent 1980) had a major objective of improving the availability of a previous generation system. The 5ESS actually replaced the 1AESS, which was AT&T's first electronic switching system (David Long, Personal Communication 2010) from 99.90 percent to 99.98 percent. In other words, that meant a change of 0.08 percent in the level of one single quality. However, that represents 80 percent of the way to perfection. Perfection costs infinity. The project took eight years, using 2,000 to 3,000 people. Assuming this is correct and realistic—even roughly so—one would need the fourth digit (xx.x0 to xx.x8 percent) to signal that the project might cost 24,000 work-years (David Long, Personal Communication 2010). COCOMO, for example, does not try to operate at this level of precision.

One organization contracted for one year of effort for 100 developers, at fixed price, and promised to deliver airplane telephone switching software with 99.999 percent availability. They contractually promised 99.999 percent without ever having achieved such a result in the entire history of the corporation, and without consulting their technical director. The CEO acknowledged that they had taken an unwarranted risk, big enough to put their company out of business. This was, of course, an organizational problem; management needed to make sure they knew what they were contracting for.

2. Experience: If one does not have experience data about the resources needed for one's technical solutions, then he or she cannot estimate the project resources.

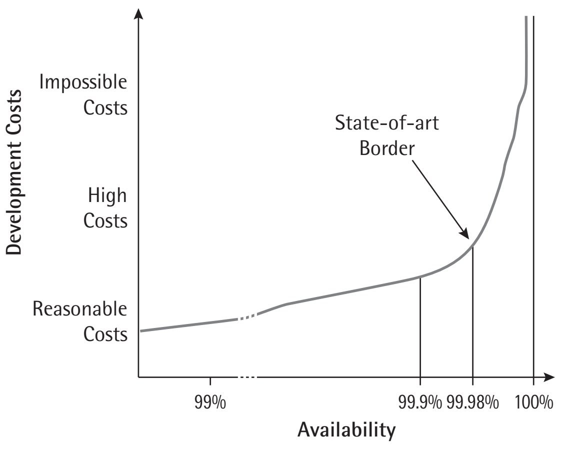
So what does it cost to develop software with 99.999 percent availability? (See Figure 2.) There are few documented instances (for example, Lucent 1980), and perhaps no instances are relevant to the current system. This is only one cost driver variable among many. People might even have a real problem of convincingly proving that the system they developed is actually at that level, and will remain at that level for, say, 40 years! Nobody knows, and nobody can be certain.

One of the problems is that by the time the experience data are available, it is likely out of date with respect to the technology to be deployed. But with no valid and useful historical experience data, nobody can know. What one can do is to avoid promising specific costs, when no one really knows what they are, and what they will become. Even if someone did know a sample of costs for the required availability levels, it takes only one other variable, such as "our new investment in the requirements process," to change the real costs by at least a factor of 2 or 3. See Figure 3 for an example.

Most people do not have such data. Most don't even know about the many factors like this that can influence costs and don't have relevant historical data about the cost factor relationships. Does this seem hopeless? Large scale estimation *is* hopeless! Once one can appreciate this, he or she can turn to solving the problem of controlling large-scale costs in a serious way.

3. Architecture: If one implements the project solutions all at once,

FIGURE 2 The high cost of perfection (Gilb 1988, 170)



without learning their costs and interactions incrementally, one cannot expect to be able to understand the results of many interactions.

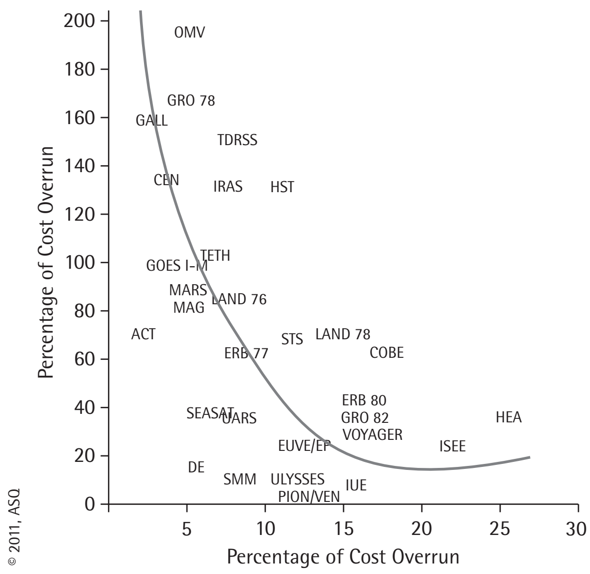
Many people are well aware that waterfall or "big bang" projects have a strong tendency to fail (MacCormack et al. 2001). But *why* do they fail?

One of the many causes of failures is that waterfall method projects are committed to doing too many things at once. It is therefore difficult to see the effects of any one design or one process, or of other single factors, on the requirements or the costs. By implementing the critical variables, such as the design or development process techniques, one at a time in small increments, then the effects can be seen more clearly. People could take steps to change unexpected and bad costs or qualities before it is too late to change, and they could better argue the need for the changes with management (see Figure 4).

One of the author's clients, Confirmit, does this very well. They measure the effect of a design on a target quality requirement, say "intuitiveness," and decide if the design is meeting its expectations. They do this in a weekly cycle. Their results are astounding cumulative quantified quality leaps, for a varied set of quality measures, which help them achieve competitive advantage (Johansen and Gilb 2005). (See Table 1.)

Single cause versus single effect is a fundamental scientific analysis principle. One cause, one effect—keep all else constant, in order to begin to understand what is going on. Although the unsimplified reality is, unfortunately, that multiple causes combine to give multiple effects. It requires some discipline (Gilb 2010b) to break things up into these small experiments, and consequently manage to build predictably realistic measurable integrated complex systems. But this step-by-step approach is the price one must pay to get some real control over costs and qualities.

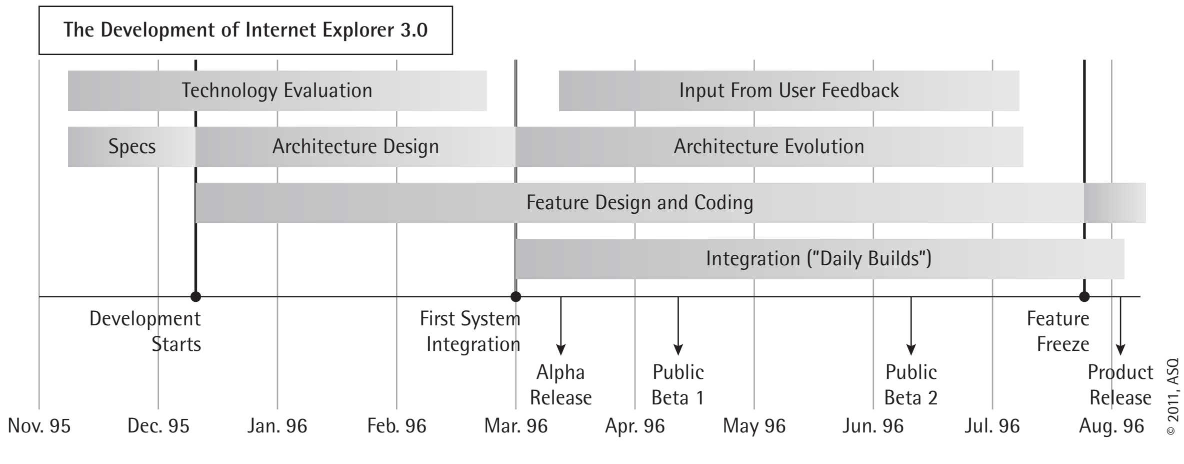
FIGURE 3 NASA Software project overrun, as a function of investment in requirements. Source: Gruehl (NASA) in (Hooks 1994)



4. Staff: If a complex and large professional project staff is an unknown set of people, or if staff changes mid-project, one cannot expect to estimate the costs for so many human variables.

Even with a track record for a defined and constant team, an organization would still have significant problems using that

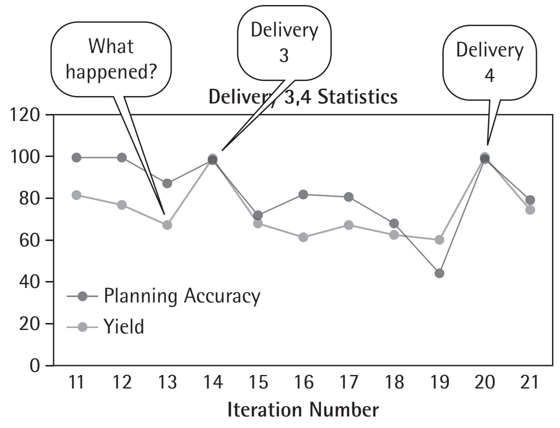
FIGURE 4 An example of a learning and change process (MacCormack et al. 2001)



information to understand their productivity and consequent time and money costs. But people normally cannot be sure who will staff their projects, or even if planned staffing can or will be carried out. And, they cannot expect that past team performance is a correct guide to future performance.

This gets even worse with offshore and outsourced projects. Honeywell discovered that top offshore staff had been swapped with less-competent staff. They realized this only by monitoring short-term numeric feedback, and finding the root cause for worse performance (Berntsen 2007) (see Figure 5). So, unknowns and unpredictable variations in people are yet another reason why it is difficult to make project cost predictions.

FIGURE 5 From Honeywell case (Berntsen 2007) where an offshore swap-out of qualified developers was caught by sensing short-term changes in the quality of work done



Measures
- **Planning Accuracy**—percent of planned work that was completed.
- **Build Yield**—percent of completed work that passed verification testing.

5. Sensitivity: If even the slightest change is made, after an "accurate" estimation, to any of the requirements, designs, or constraints, then the estimate might need to be changed radically. And, one probably will not have the information necessary to do it, nor even the insight needed to do it.

Consider the previous example (Principle 1) where a 0.08 percent increase in availability for the software of the AT&T 5ESS project cost eight years and took the efforts of thousands of developers. The point being made then was that system qualities are major cost drivers. But there is an additional point that these cost drivers can be very sensitive to apparently small numeric changes in requirements, or to slight numeric changes in actually delivered quality. For example, 0.08 percent more than required might cost a bundle now, and 0.08 percent less than required might cost a bundle later.

Costs can be sensitive to drivers other than performance requirements and quality requirements. They can also be sensitive to resource constraints, such as people, time, money (for development and for maintenance), legal constraints (national sensitivity to personal data, for example), policy constraints ("do no evil"), and a large number of factors. If these cost-driving factors are not clearly specified, and are not really followed up on in practice, then the real costs might vary widely, compared to expectations, as a result. The real costs might be surprisingly higher costs initially, or they may come as a big surprise, due to unforeseen factors later—the true long-term costs.

TABLE 1 A simplified version of the impact estimation (IE) table [3,2] for Evo Step 9, "Recoding" of the "Market Information Recoding" design

Notice the definitions for the requirements and costs. The Planguage keyed icon "<->" means "from baseline to target value." Step 9 alone moved the Productivity value to 27 minutes, or 95 percent of the way to the target level (Johansen and Gilb 2005).

	Design Idea: Step 9—Recording			
Requirements	**Estimated Scale Impact**	**Estimated Percent Impact**	**Actual Scale Impact**	**Actual Percent Impact**
Objectives				
Usability. Productivity 65 <-> 25 minutes Past: 65 minutes. Tolerable: 35 minutes. Goal: 25 minutes.	65 – 20 = 45 minutes	50%	65 – 38 = 27 minutes	95%
Resources				
Development Cost 0 <-> 110 days	4 days	3.64%	4 days	3.64%

The unpleasant fact is that even the best of organizations are embarrassingly bad at clear and complete specification of requirements. They do not try very hard to be complete, and they do not seem to know how to be clear on the critical few top-level requirements, let alone on details that might be significant enough to affect costs significantly (Gilb 2005; Gilb 2008a) (see Figure 6).

To summarize so far, the risk principles, discussed previously, are one way of saying that any attempt to estimate costs and timing, based on current requirements practices, and even on vastly more clear and complete requirements practices, are doomed to failure. Managers should always regard any such estimates as highly suspicious.

In fact, if the staff providing the estimates are themselves not explicitly aware of this fact, their competence is dubious. They ought to give fair warning, like this for example:

CAVEAT: The estimates cannot be trusted, even regarding their order of magnitude. There are too many unknown and unknowable factors that can significantly affect the results.

FIGURE 6 Real case study example (Gilb 2008a) of the primary objectives for a project that took 10 years before delivery of any of these objectives, and cost more than $160,000,000

Primary Objectives for a Project

1. Central to the corporation's business strategy is to be the world's premier integrated <domain> service provider.
2. Will provide a much more efficient user experience.
3. Dramatically scale back the time frequently needed after the last data are acquired to time align, depth correct, splice, merge, recomputed, and/or do whatever else is needed to generate the desired products.
4. Make the system much easier to understand and use than has been the case for the previous system.
5. A primary goal is to provide a much more productive systems development environment than was previously the case.
6. Will provide a richer set of functionality for supporting next-generation logging tools and applications.
7. Robustness is an essential system requirement.
8. Major improvements in data quality over current practices.

The complete lack of measurable precision in these primary project objectives is, in the author's view, the primary reason for the time delay and costs. Most managers allow such things to happen on most projects, according to the author's decades-long world-wide experience. They lose control of costs immediately when primary critical project objectives are unclear. Notice that all objectives refer to qualities (of an existing system).

The estimates could be used as a framework budget. But one would have to evolve the system in small steps, and learn from experience what the real costs, real requirements, and real designs are. The only way to be reasonably sure of the (money) costs and the (effort) deadlines would be to apply redesign-to-cost adjustments as the project progresses (Mills 1980).

If the staff did this, then they have given management fair warning, and also a prescription for success. They have also understood the control principles in the second half of this article. The estimation reality then becomes:

- Management can decide, based on the projected value of the system, how much they can spend on the project (that is, their budget).
- They can decide, based on the market situation, *when* they want the stakeholder value to be delivered (that is, the deadlines).
- They can then design the project organization, to deliver the value, when they want it, at a cost they find affordable.
- If even the simplest and smallest (weeks or months) attempts to deliver value within satisfactory time and cost fails, then management has an incompetent team, an ineffective process, or an impossible project, and they should take that as a warning to stop or change.

This is, of course, a big change to the way IT or software projects are managed. The beauty of the control principles are that they do not take a long time to prove they work in practice.

When Estimates Might Work

There are times when making estimates might work well enough for practical purposes, or might seem to work:

- If effort on the project ceases, when deadlines and budgets are used up.
- If the qualities and performance of the system are not yet at required levels, but people have no better expectations, and they are prepared to improve qualities over time to reach satisfactory levels.
- If staff/contractors are highly motivated to NOT exceed budgets and deadlines.

Stopping project effort, when dubiously estimated resources are used up, does not prove the estimates were ever correct. It just proves that one can stop and deliver something without being recognized as a clear failure. Usually one can do that (deliver something useful) with no effort or cost whatsoever, by using a previous or old system. So, any effort spent improving the older system will often be appreciated—especially if some improvements have been made visible early.

THE CONTROL PRINCIPLES

The previous discussion has tried to establish that there is no reasonable way to get useful (credible) estimates for nontrivial software projects. If the real final project (or product delivery and service costs) costs destroy profit margins, or destroy the planned return on investment or management reputation, they are not useful (Collins and Bicknell 1998; Craig and Brooks 2006).

However, assuming one is still interested in what to do about the estimation problem, the following five control principles (principles 6 to 10 below) offer some practical solutions. They can be simplified into a four-step process as follows:

1) Do something of value in a short time.
2) Measure values and costs.
3) Adjust what to do next, if necessary.
4) Repeat until there is no longer value for money.

See also (Denning 2010) for further discussion on this.

Advantages with this method:

1) Too great an amount of time or money cannot be wasted before one realizes that he or she has false ideas.
2) Value is delivered early, and it keeps people happy.
3) The organization is forced to think about the *whole* system, including *people* (not just the code).
4) The organization is destined to see the true costs of delivering value—not just the code costs.
5) One can learn a general method that he or she can apply for the rest of his or her career.

Disadvantages with this method:

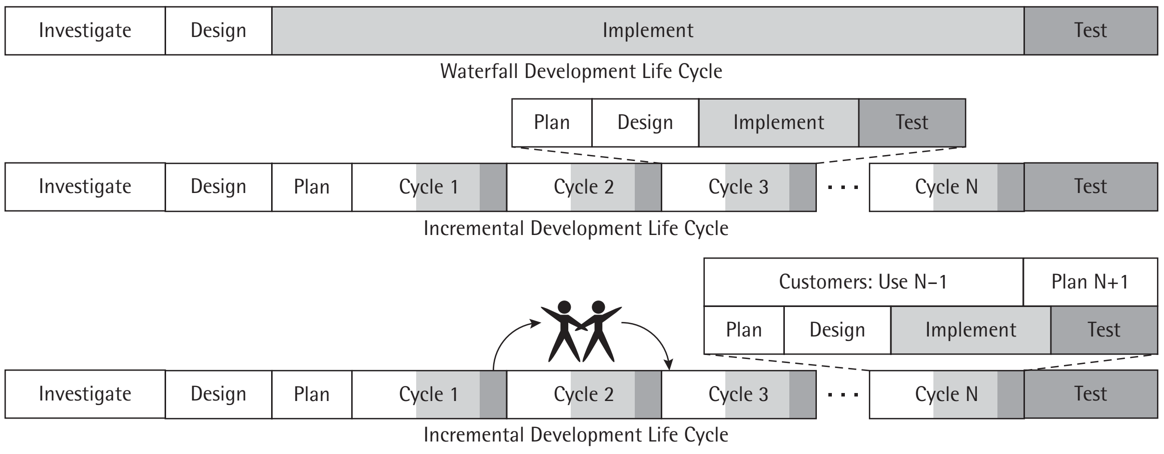
1) One cannot hide his or her ignorance any longer.
2) One might have to do something not taught at school, or not taught in textbooks.
3) There will always be people who criticize anything different or new.
4) One cannot continue to hide any lack of ability to produce results inside a multiyear delayed project.

The Control Principles (6 to 10 of the 10 Estimation Principles).

6. Learn small: Carry out projects in small increments of delivering requirements, so one can measure results and costs against (short-term) estimates.

All software projects can be broken into early, small increments—not merely increments of building, but more importantly, increments of delivering value to one's stakeholders (Gilb 2010b) (see Figure 7).

FIGURE 7 Concepts of small delivery cycles with stakeholder feedback, from HP, a client who uses the Evolutionary Project Management (Evo) method (Cotton 1996)



Almost all projects can, if they really want to, start delivering some value to some stakeholders, next week and every week thereafter. But most project managers don't even try, and don't even try to learn how.

Leaving aside "*how* to decompose" for a moment (Gilb 2008b; Gilb 2010a; 2010b), if one can deliver early stakeholder value, then he or she gets control over "value for money."

- Select the highest available deliverable value, each step.
- Deliver it to the most critical stakeholder.
- *Prove* that the designs actually work.
- Get some sense of the time needed and monetary costs.

These steps should usually be about one week at a time. Assuming one can deliver reasonable value for effort spent, week after week, surprising things then happen:

- People cease to care about the conventional deadlines.
- People cease to ask for estimates of the monetary budget.
- People are strongly encouraged to keep on going, until the value to be delivered is less than the costs.
- The project ends up delivering far more real value than other projects do, well before the end of the project (without this approach a conventional project deadline would have been set, and would also have been overrun).
- Management shifts focus from, the budget and the costs, to return on investment (ROI).

Consider the 1.1.1.1.1.1 method or the Unity Method (Gilb 2010b):

Plan, in one week
To deliver at least 1 percent
Of at least one requirement
To at least one real stakeholder
Using at least one design idea
On at least one function of the system.

It is amazing the practical power of this simple idea of unity. If one really tries, and management persists with encouragement and support, it almost always works. In one "outside-the-box" example, 25 aircraft projects at Boeing (then McDonnell Douglas) used this 1.1.1.1.1.1 method in practice, identifying and getting approval for the small high-value delivery steps (Gilb 2002). No exceptions. It is not a principle limited to software, though most experience of its use is in that area.

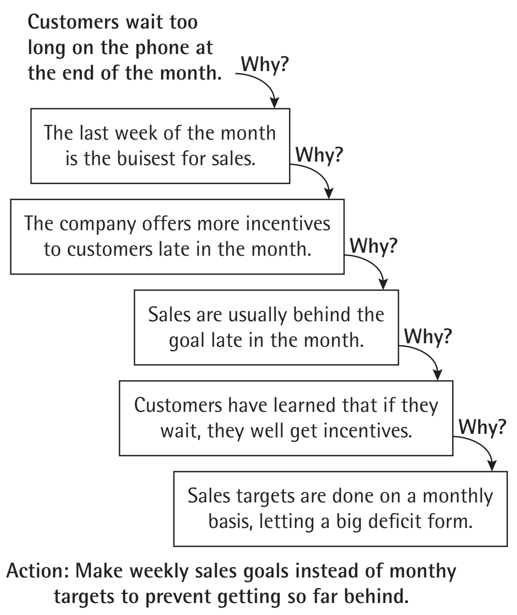
7. Learn root: If incremental costs for a given requirement level (and its proposed designs) deviate negatively from estimates—analyze the deviation's root cause, and change anything about the coming increments that might get things back on track.

The set of designs needed to deliver a demanding quality level can usually be implemented one design at a time. One should also make an estimate of the expected impacts for the highest-contribution-to-requirement-levels design options and implement them, with the *highest* estimated impact *first*. One should also estimate the expected development cost for each design increment, and choose the one that gives the best impact on quality, in relation to its *estimated* costs (the best ROI).

On deployment, one must try to measure both the actual impact, to make sure it is roughly as estimated, and note the actual costs. Keep track of the real costs, and note the real initial development cost of getting to the required quality levels. It is essential to learn as much as possible, as early as possible, from implementation and deployment; there are likely to be some surprises about the actual costs. One process for learning is to seek the root cause (Goldratt 2008) of the deviation from one's estimates. If one can find the root cause, and therefore try something to avoid it, he or she can reduce costs and improve quality levels (and move toward estimates). A simple strategy for root cause thinking that works well in practice is asking "Why?" The Japanese "Five Whys" method (Wikipedia 2010) (see Figure 8) hints at the need for continuing to ask this question of the resulting answer until one hits an answer that is able to be dealt with effectively.

Managers are very frequently at the wrong level of "Why?" in understanding a problem; root cause analysis can help. One instance of this was reported by David Long (Personal Communication 2010) referring to a very large software project, AT&T Switching 5ESS. He wrote: "But, from what I saw, the greatest improvements came from a small amount of work that was done as

FIGURE 8 Simple example of applying the 5 Whys (Velaction 2010)


Customers wait too long on the phone at the end of the month.
Why?
The last week of the month is the buisest for sales.
Why?
The company offers more incentives to customers late in the month.
Why?
Sales are usually behind the goal late in the month.
Why?
Customers have learned that if they wait, they well get incentives.
Why?
Sales targets are done on a monthly basis, letting a big deficit form.
Action: Make weekly sales goals instead of monthy targets to prevent getting so far behind.

the result of specific root cause analysis performed on specific outages."

One can conclude that there is potentially considerable cost reduction leverage to be had from root cause analysis. The main leverage is in solving the critical few problems that are frequent and damaging. However, how can one factor such processes into an initial cost calculation? One cannot. It must be done incrementally, and then maybe one can reduce costs, by redesign, to the level some cost-optimist has estimated.

8. Prioritize critical: One will have to prioritize the most critical requirements and constraints: There is no guarantee one can achieve them all. Deliver "high-value for resources-used" first.

A time or cost estimate, once it becomes a deadline or a budget, is a "limited resource." One of the smartest ways to deal with limited resources is intelligent prioritization (Gilb and Maier 2005). Instead of implementing all of the designs in a big-bang manner, and hoping to meet all the requirements and resource (time and cost) estimates, try delivering the project value a little bit at a time, and see how each of the designs actually works, and what they actually cost. "A little bit at a time" should be interpreted as delivering evolutionary steps of approximately 2 percent of the overall project timescales (say weekly, fortnightly, or monthly cycles), and hopefully about 2 percent of the project financial budget.

Planguage's impact estimation (IE) method (Gilb 1988; Gilb 2005) is helpful in identifying the designs that give the estimated best stakeholder value for their costs. People should implement designs incrementally so they can get acceptable feedback on costs and value delivery (in terms of meeting requirements). With a bit of luck, stakeholders then receive interesting value very early, and what works and what doesn't is learned in practice. If a deadline is hit, or if the budget runs out, then at least there is still a complete live real system with delivered high value. In such situations, the whole concept of the project-level estimates, the deadlines, and the budgets is not so important any more.

In the Cleanroom Method, developed by IBM's Harlan Mills (1980) they reported:

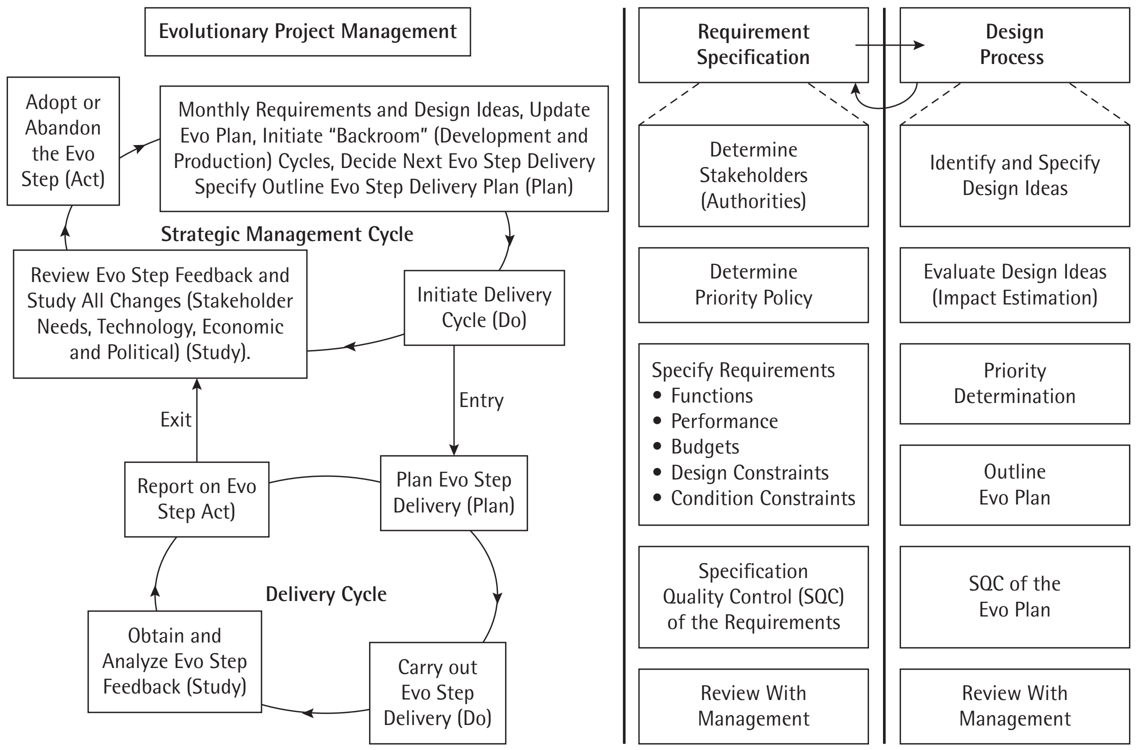
> "Software engineering began to emerge in FSD" (IBM Federal Systems Division, from 1996 a part of Lockheed Martin Marietta) "some 10 years ago [about 1970] in a continuing evolution that is still underway:
>
> - Ten years ago general management expected the worst from software projects—cost overruns, late deliveries, unreliable and incomplete software.
> - Today [in other words, in 1980], management has learned to expect on-time, within budget, deliveries of high-quality software. A Navy helicopter ship system, called LAMPS, provides a recent example. LAMPS software was a four-year project of over 200 person-years of effort, developing over three million, and integrating over seven million words of program and data for eight different processors distributed between a helicopter and a ship in 45 incremental deliveries. Every one of those deliveries was on time and under budget.
> - A more extended example can be found in the NASA space program.
> - Where in the past 10 years [from 1070 to 1980], FSD has managed some 7,000 person-years of software development, developing and integrating over 100 million bytes of program and data for ground and space processors in over a dozen projects.
> - There were few late or overrun deliveries in that decade, and none at all in the past four years."

Harlan Mills told the author that they had to solve the persistent problem of cost overruns, in relation to the contracted estimates, because the government was getting smarter and using fixed-price contracts (as opposed to cost plus). If the project ran over, IBM lost its own profit. If IBM did not find a better way, it could just as well leave the business. Notice the "45 deliveries" Mills cites. That means 2 percent delivery cycles. IBM was using intelligent feedback and learning. It actually had very sophisticated estimation technology based on a thorough collection of experiences (Walston and Felix 1977). But this did not give IBM the accuracy it needed to avoid losing money. Say it had a 40 percent profit margin, and it could be wrong by 40 percent to 200 percent (the NASA range, see Figure 3). IBM would still lose money on most contracts. So, it had to compensate for its estimation inaccuracy by incremental feedback and necessary change, to come in "on time and under budget every time." Mills' colleague, Quinnan, describes the process of intelligent dynamic prioritization in detail in (Mills 1980).

A thorough explanation of systems development processes that can help achieve intelligent prioritization is found in (Gilb 2005) (see also Figure 9). Nothing less will suffice for large and complex systems (that is, for projects involving hundreds of staff, years of effort, and sometimes $100 million or more in cost). For smaller systems (small teams over a few months), several organizations have made good simple practical adaptations (Upadhyayula 2001; Johansen and Gilb 2005) of the essential "Evo" development process ideas (Evo itself predating later agile methods) (Denning 2010, 129; Gilb 2010c).

The agile community has adopted small iterative cycles (Gilb 2010c), but they have failed to adopt the notion of measurement of value and quality, which is essential for larger projects and some smaller ones. Without explicit quality requirement and design impact metrics, the value prioritization is ambiguous and too subjective (Gilb 2010c; Gilb and Brodie 2010).

FIGURE 9 Priority management as an iterative process (Gilb and Maier 2005)


Evolutionary Project Management
Adopt or Abandon the Evo Step (Act)
Monthly Requirements and Design Ideas, Update Evo Plan, Initiate "Backroom" (Development and Production) Cycles, Decide Next Evo Step Delivery Specify Outline Evo Step Delivery Plan (Plan)
Strategic Management Cycle
Review Evo Step Feedback and Study All Changes (Stakeholder Needs, Technology, Economic and Political) (Study).
Initiate Delivery Cycle (Do)
Entry
Exit
Report on Evo Step Act)
Plan Evo Step Delivery (Plan)
Delivery Cycle
Obtain and Analyze Evo Step Feedback (Study)
Carry out Evo Step Delivery (Do)
Requirement Specification
Design Process
Determine Stakeholders (Authorities)
Identify and Specify Design Ideas
Determine Priority Policy
Evaluate Design Ideas (Impact Estimation)
Specify Requirements
• Functions
• Performance
• Budgets
• Design Constraints
• Condition Constraints
Priority Determination
Outline Evo Plan
Specification Quality Control (SQC) of the Requirements
SQC of the Evo Plan
Review With Management
Review With Management

The left-hand of the diagram shows the strategic management cycle and the delivery cycle within Evolutionary Project Management (Evo) (the development cycle and production cycle are not shown). Within the strategic management cycle, the system requirements and design ideas can be updated to reflect feedback, and any changes that have occurred. Within the delivery cycle, actual delivery of the Evo step occurs. The right-hand of the diagram shows the main sub-processes of requirement specification and the design process (Gilb 2005). Note impact estimation and priority determination are within the design process.

9. Risk Fast: Implement the design ideas with the "highest value with regard to cost and risk" early.

One can use the IE method (Gilb 2005) to identify the designs with highest value with regard to cost and risk, and then try them out first.

Looking at the design ideas in Figure 10 (from real case at Ericsson), defined only at the "Gist" (summary) level (there *was* in the real example more detail to the definition, but this level is OK for the purposes here), which of these designs is high risk? There is no documentation here as to what experience would lead one to expect of costs or results, so there is no perceivable difference. Unfortunately, too many design specifications are at this level—no information about interesting quality and cost differences, no estimates for costs, no estimates for expected impacts on requirements, and no estimates of the certainty of the estimates.

FIGURE 10 Brief description of a simple real example of some design ideas to improve learning time (Gilb 2005, 267)

Design Ideas

Online Support: Gist: Provide an optional alternative user interface, with the users' task information for defined task(s) embedded into in.

Online Help: Gist: Integrate the users' task information for defined task(s) into the user interface as a "Help" facility.

Picture Handbook: Gist: Produce a radically changed handbook that uses pictures and concrete examples to *instruct*, without the need for *any* other text.

Access Index: Gist: Make detailed *keyword indexes*, using *experience* from *at least 10* real users learning to carry out the defined task(s). What do *they* want to look things up under?

IE documents the error margins and uncertainty, to better understand risk. To illustrate how IE achieves this, here is a real example. The main requirement is called "Learning," and the numerically specified requirement was to reduce the learning time from 60 minutes to 10 minutes (see Figure 11).

- **Scale impact**: The first set of estimates are a best guess as to the result if each design was implemented individually and incrementally. "Online Help" is estimated to get the organization to the required 10 minutes.
- **Scale uncertainty**: The scale impacts are just estimates, so the organization must assess the likely best-case and worst-case *range* of estimates. This is one means of expressing that there is a risk that the real result will NOT be the initial estimate. In this case, the ± five minutes for "Online Help" means the organization thinks the worst they could get is a result of 15 minutes (10 + 5) and the best is five minutes (10 - 5).
- **Percentage impact:** A way of expressing these same estimates in direct relation to the "Learning" requirement level (where 10 minutes = 100 percent of the objective). This is not important here. The percentage impact is used as a common currency to make comparisons requirements that we might well be looking at simultaneously to make a risk decision. But not in this simplified example.

Now based on the scale impact and the scale uncertainty, which one of the four designs is the "smartest" one to try out first? From the author's perspective, the design "Online Help" looks good, and "Picture Handbook"

FIGURE 11 An impact estimation table showing the impact of the design ideas in Figure 10 on the Learning objective (Gilb 2005, 267)

	Online Support	Online Help	Picture Handbook	Online Help + Access Index
Learning 60 minutes <–> 10 minutes				
Scale Impact	5 min.	10 min.	30 min.	8 min.
Scale Uncertainty	±3 min.	±5 min.	±10 min.	±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

looks like a loser. If "Online Help" succeeds, one would not need the other design ideas. So the IE table provides a warning that the "Picture Handbook" design has a high risk of NOT providing the desired result. The organization is primarily interested in the risk that it will cost them more than they have estimated, or cost more than what they can tolerate, to get the result. So they have to also consider some additional information. How good is the estimate of the impact? Who made it, and on what basis? It is rare to find designers for software systems documenting the "why" and "who" of estimates, let alone the estimate of impact on a numeric quality requirement, like "Learning."

Evidence/Source

The evidence (on what basis, the why) and the source (person or document for the evidence, the who) are document as best as possible. Further, based on that evidence and source information, a credibility score is assigned (0.0 worthless to 1.0 perfect) (Gilb 2005). In this case, "Online Help" comes out best (0.8) so one can stick with the belief that it will provide the best result. But, that does not settle the question of the costs and their uncertainty.

Development cost

One can make an estimate of the development cost (and/or any costs aspects of interest, such as development duration). "Online Help" is estimated to cost 25,000 monetary units.

Credibility-adjusted performance to cost ratio

When one modifies the costs with the credibility factor—a rough measure of how well one can trust the design impacts and costs based on history (evidence and source), it becomes even more convincing that one should stick with "Online Help" as the design to try first (3.2 being the largest value for money, credibility adjusted). Or perhaps one could make a ± estimate for the costs, and consider how wide the range of cost estimates is (that is, how risky), and what the worst case equates to (estimate plus the highest cost border).

This might seem like a lot of bureaucracy, but it is a lot cheaper than just diving in and implementing unevaluated designs, and then failing, as software people so often do. It is, in fact, a reasonable fact-based reasoning or logical process that one should try to do. If one consistently chooses the safest bets, that is, the iterative implementation options that promise, based on good evidence, to deliver best requirement value for costs, then costs will be under better control. However, this cannot be done all at once. One must use feedback and also learn to estimate better, and one must provide better evidence and better sources. In other words, an organization must learn to control costs during the project.

Sometimes doing high risk first is useful

There is a related principle, which may *seem* contradictory, that given several designs to be implemented—one should not do the potentially high-performance to cost design early, but the high risk design instead. This assumes that there is high value at low cost *estimated*, but that the ± uncertainty is very wide, for example, 80 percent ±70 percent for a quality or a cost budget. In that case there is an argument that one should find out whether the design is as promising as he or she optimistically would like it to be, or not. There is a high value in knowing the reality. In this case, the stakeholder is the system architect, and the *value* is the value to the architect of validating the true impact and costs of their architecture, in good time, and being able to retreat, should it be falsely overoptimistic.

10. Apply now: Learn early, learn often, learn well, and apply the learning to the current project.

The 10th control principle is implied by, and discussed in connection with, the other principles. To get control over costs and budgets, one must get to the truth of the cost of meeting project requirements. One must humbly recognize that no one else is certain of the costs.

Start learning the true costs of the designs for meeting project requirements as early as possible. One should start learning for real, the second week of any project. Meetings and opinions just can't beat reality. The pace of learning cycles should be weekly until all requirements are delivered, or until resources run out. Longer two-weekly or even monthly cycles are also practical.

SUMMARY

The world of software is complex enough in itself—some say software projects are the most complex projects humans undertake. However, software is always part of a larger system of machines, people, laws, markets, and politics. So understanding the real costs accurately for developing initial solutions, to deliver competitive levels of performance and quality, is near impossible up front. Cost estimation requires far more than understanding code, user stories, function points, and use cases. People will make estimates, and with some luck might get the order of magnitude right. But management would be foolish to believe these estimates are sufficiently reliable to bet their own career on them.

Long-Term and Total Costs

Understanding *initial* development costs is tricky enough, but understanding long-term operational and maintenance costs—which usually dwarf initial development costs—is even more difficult, as it is dealing with a more unmeasurable and unpredictable future. Something more than the suggested principles apply to this problem. One is, for example, talking about how to engineer long-term cost-drivers into the system, initially and gradually (Grimstad, Jørgensen, and Moløkken-Østvold 2006). So management's strategic planning needs to take into account these principles:

- Management needs to base their planning on the cost they are prepared to pay to deliver specified value (as specified in quantified requirements). This is keeping an eye on the financial value, or profitability (or maximum profitable cost (MPC)).
- Management needs to also decide the time value of delivering specific system value levels (such as 99.99 percent availability within 12 months) (or high value delivery timing (HVDT)).
- Management then needs to make sure their projects are done one step at a time (2 percent steps of the MPC and/or the HVDT). Management needs to make sure projects are organized to deliver the highest possible value as early as possible, and that projects learn what is real very quickly.

Acknowledgement

The author would like to thank Lindsey Brodie, University of Middlesex, London, UK, for her help in editing this article.

References

Berntsen, J. 2007. *A case study in globally distributed lean software development.* ICSPI 2007 Conference. Available at http://www.gilb.com/tiki-download_file.php?fileId=432.

Boehm, B., C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece. 2000. *Software cost estimation with Cocomo* II. Englewood Cliffs, NJ: Prentice Hall. Available at: http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html.

Collins, T., with D. Bicknell. 1998. *CRASH: Learning from the world's worst computer disasters.* London: Simon and Schuster Ltd.

Craig, D., and R. Brooks. 2006. *Plundering the Public Sector: How New Labour are letting consultants run off with £70 billion of our money.* London: Constable & Robinson Ltd.

Cotton, T. 1996. Evolutionary fusion: A customer-orientated incremental life cycle for fusion. *Hewlett-Packard Journal* 4:25-38. Available at: http://www.hpl.hp.com/hpjournal/96aug/aug96a3.htm.

David Long 2010. Email discussion with the author about Lucent switching system history.

See also Reference Lucent 1980.

Denning, S. 2010. *The leaders guide to radical management, reinventing the workplace for the 21st century.* San Francisco: Jossey-Bass/Wiley.

Gilb, T. 1988. *Principles of software engineering management.* Boston: Addison-Wesley.

Gilb, T. 2002. *DAC case studies: Management objectives, QC and a little about 25 Evo projects for aircraft engineering.* Available at: http://www.gilb.com/tiki-download_file.php?fileId=254 [Accessed 27.Dec.2010].

Gilb, T. 2005. *Competitive engineering: A handbook for systems engineering, requirements engineering, and software engineering, using planguage.* Oxford: Elsevier Butterworth-Heinemann. Available at: http://www.gilb.com/tiki-download_file.php?fileId=77 and http://www.gilb.com/tiki-download_file.php?fi.

Gilb, T. 2006. *No cure no pay.* Available at: http://www.gilb.com/tiki-download_file.php?fileId=38.

Gilb, T. 2008a. *Top level critical project objectives: How to quantify and control key objectives at all levels of the project.* Available at: http://www.gilb.com/tiki-download_file.php?fileId=180.

Gilb, T. 2008b. Decomposition of projects: How to design small incremental steps. In *Proceedings of INCOSE 2008.* Available at: http://www.gilb.com/tiki-download_file.php?fileId=41.

Gilb, T. 2010a. Value planning slides for Scrum product owners. Available at: http://www.gilb.com/tiki-download_file.php?fileId=353.

Gilb, T. 2010b. The 111111 or Unity Method for Decomposition, Presented at the 2010 Smidig (Agile) Conference, Oslo. Available at: http://www.gilb.com/tiki-download_file.php?fileId=350.

Gilb, T. 2010c. Value-Driven Development Principles and Values – Agility is the Tool, Not the Master. *Agile Record*, July 2010, 3. Also available at: http://www.gilb.com/tiki-download_file.php?fileId=431.

Gilb, T., and L. Brodie. 2010. Principles and values – Agility is the tool, not the master, Part 2: Values for Value. *Agile Record* (October). Also available at: http://www.gilb.com/tiki-download_file.php?fileId=448.

Gilb, T., and M. W. Maier. 2005. Managing priorities: A key to systematic decision-making. In *Proceedings of INCOSE 2005.* Available at: http://www.gilb.com/tiki-download_file.php?fileId=60.

Goldratt, E. M. 2008. *The choice.* Great Barrington, MA: North River Press.

Grimstad, S., M. Jørgensen, and K. Moløkken-Østvold. 2006. *Software effort estimation terminology: The tower of Babel.* Available at: http://simula.no/research/se/publications/Grimstad.2006.1/simula_pdf_file.

Hooks, I. 1994. *Guide for managing & writing requirements.* Boerne, TX: Compliance Automation Inc.

Johansen, T., and T. Gilb. 2005. 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. Available at: http://www.gilb.com/tiki-download_file.php?fileId=32.

Jørgensen, M. 2005. Practical guidelines for expert-judgment-based software effort estimation. *IEEE Software* (May/June). See also 2008 slides, Software Development Effort Estimation: Why it fails and how to improve it. Available at: http://simula.no/research/se/publications/Simula.SE.375.

Lucent. 1980. Lucent AT&T 5ESS case study. See http://www.mdavidlong.com/resume_m_david_long.pdf.

"5ESS system reliability of 99.9999%"

MacCormack, A., M. Iansiti, and R. Verganti. 2001. Product-development practices that work: How Internet companies build software. *MIT Sloan Management Review* 42, issue 2 (Winter).

Mills, H. 1980. The management of software engineering: part 1: principles of software engineering. *IBM Systems Journal* 19, issue 4 (Dec.):414-420.

Upadhyayula, S. 2001. Rapid and flexible product development: an analysis of software projects at Hewlett Packard and Agilent. Masters thesis, Massachusetts Institute of Technology. Available at: http://www.gilb.com/tiki-download_file.php?fileId=65.

Velaction. 2010. *Lean term: 5 Whys*. Available at: http://www.velaction.com/5-whys/.

Walston, C. E., and C. P. Felix. 1977. A method of programming measurement and estimation. *IBM Systems Journal* 1:54-73. Available at: http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=5288519.

Wikipedia. 2010. *5 Whys*. Available at: http://en.wikipedia.org/wiki/5_Whys.

Biography

Tom Gilb has been an independent consultant, teacher, and author since 1960. He mainly works with multinational clients helping improve their organizations, and their systems engineering methods. Gilb's latest book is *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage* (2005). His other books include *Software Inspection* co-authored with Dorothy Graham (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. Gilb's key interests include business metrics, evolutionary delivery, and further development of his planning language, Planguage. He can be reached at tom@gilb.com or at http://www.gilb.com.