Estimation or Control?

- ***A Paradigm Shift in Agile and Lean Directions.***

- "*Accurate* estimation is impossible for complex technical projects, but keeping to agreed budgets, and deadlines is possible, using feedback and change."

By © Tom@Gilb.com www.Gilb.com (where a copy of this draft is stored)

Version: Initial draft 25 July 2010

Introduction

Accurate estimation, of time and costs, for complex systems and software projects is seemingly impossible to guarantee [9,10, 12, 25]. 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, but as a result - delivering systems of unacceptably low quality.

The main idea of this paper is that there is a constructive alternative to unsatisfactory estimation processes.

- You use 'process control' (do a little, learn quickly, change quickly) to quickly and continuously tune anything and everything about your project, so that prioritized budgets (like time to market, money, human resources) can be met.
- You consciously sacrifice less-holy things for your more-holy things

We are better off *stipulating* reasonable resource constraints (deadlines and cost budgets) and then learning to live within them. This is acceptable as long as we get our highest priorities satisfied when resources run out. The rest is by definition, marginal.

Why we cannot estimate project costs accurately because:

- We do not *define requirements* well enough to estimate their costs
- We do not specify our designs (aka strategies, architecture) needed to satisfy clear and complete requirements, well enough to know the cost consequences
- We do not collect or have access to *experience data* that would allow us to estimate costs, for our well specified designs, even if we did have clear requirements
- Even if we did have *historic* data, for the costs of meeting clear requirements, it would probably not help much because our new *project would be different* in some decisive way.

We do *not really* need time and cost estimates:

- They 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. But estimates do not, in fact, prevent overruns or assure value.
- What management needs is delivery of some critical system requirements in a predictable time frame
- And they need to be sure that the project will be profitable, and not embarrass them with unexpected losses.

- This paper describes an alternative way to achieve these management needs.

Principles of Resource Control - Summary

The Risk Principles

1. DRIVERS: If you have not specified all critical performance and quality levels numerically – you cannot estimate project resources.
2. EXPERIENCE: If you do not have historic data about the resources needed for your technical solutions - you cannot estimate project resources.
3. ARCHITECTURE: If you do your project solutions all at once without learning their costs and interactions – you cannot expect to be able to estimate the results.
4. STAFF: If a complex and large professional project staff is an unknown set of people, or changes in mid project – you cannot expect to estimate the costs.
5. SENSITIVITY: If even the slightest change is made after an 'accurate' estimation, to any requirements, designs or constraints – then the estimate might

need to be changed radically. And – you probably will not have the information necessary to do it, nor the insight that you need to do it.

The Control Principles

6. LEARN SMALL: Do projects in small increments of delivering requirements – so you can measure results and costs against estimates.

7. LEARN ROOT: If incremental costs for a given requirement deviate negatively from estimates – analyze the root cause, and change anything about next increments that you believe might get you back on track.

8. PRIORITIZE CRITICAL: You will have to prioritize your most critical requirements and constraints: there is no guarantee you can achieve them all. Deliver high-value for resources-used *first*.

9. RISK FAST: You should probably implement the highest value with regard to cost and risk design ideas early.

10. APPLY NOW: Learn early, learn often, learn well, and apply the learning to your current project.

Some Constructive suggestions to people who want estimates.

- Stipulate one or more useful deadlines from your 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.
- Stipulate the value to you of reaching each major (top 10) requirement. Make an outsourcing contract, and pay part of that value, only when that requirement is successfully delivered.
- Do business with suppliers who consistently deliver value for money. Don't waste your money on suppliers who make excuses. 'No cure, no pay' [8] is one way to motivate suppliers to give you value for money – otherwise their motivation is to keep billing you forever [10].
-

Principles of Resource Control - in more detail

For those of you who like short papers. I'm done; using the summary above.

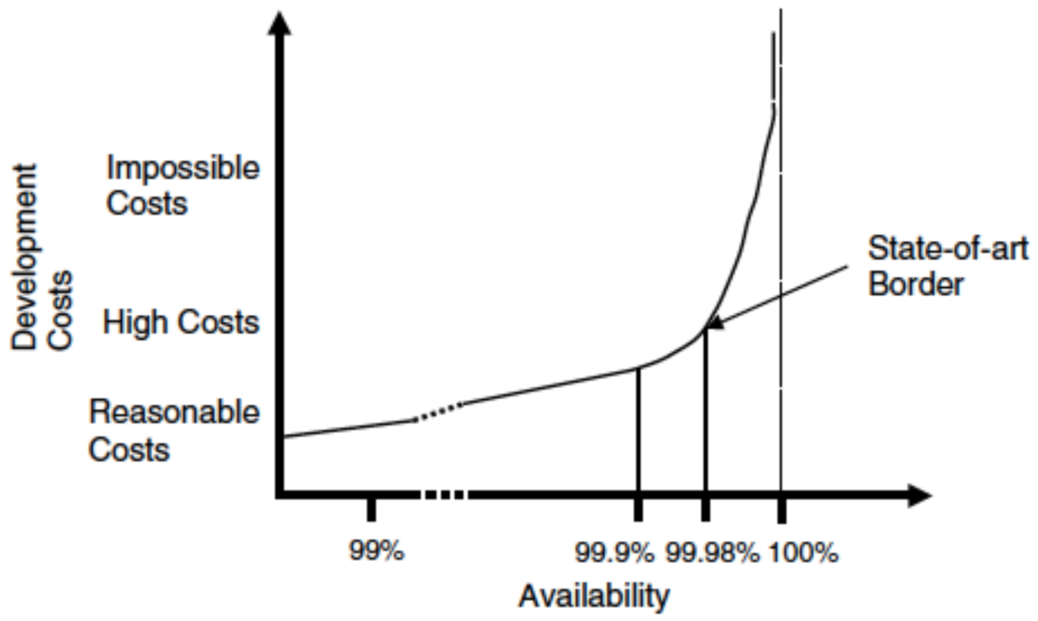
However some of you might like more explanation, more detail, more references, and convincing argument for yourself or others. So this follows here.

The Risk Principles – or 'why you really cannot expect estimates to be correct, accurate, reliable, true'.

1. DRIVERS: If you have not specified all critical performance and quality levels numerically – you cannot estimate project resources.

Costs are a function of the designs you deploy, in order to meet the requirements you need to meet. In particular they are a function of the performance and quality requirements. They are not so sensitive to functions or size – although there is a relationship there too.

It has long been understood, for example COCOMO [4], that there are a large number (dozens) of software cost drivers, such as availability. In one case [11] the 5ESS project, had a major objective of improving availability of a previous generation system (The 5ESS actually replaced the 1AESS, which was AT&T's first electronic (program controlled) switching system. [21]) from 99.90% to (5ESS) 99.98%. A change of 00.08% in one quality. But, 80% of the way to perfection, which costs infinity. The project took 8 years, using 2 to 3,000 people. Now assuming this is correct and realistic – even roughly so – we would need the 4th digit (xx.xo to xx.x8%) to signal that our project might cost 24,000 work-years. [21]



Source: CE book [3, also 2] page 170.

One of my clients made the mistake, I discovered, of contracting for one year of effort, for 100 developers, fixed price, and promised to deliver airplane telephone switching software with 99.999% availability. They did so without ever having achieved such a result in the entire 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, to make sure that they knew what they were contracting for.

2. EXPERIENCE: If you do not have historic data about the resources needed for your technical solutions - you cannot estimate project resources.

So what does it cost to develop software with 99.999% availability? [11b]

There are few documented instances, and perhaps no instances are relevant to our current system. This is only one cost driver variable among many. We might even have a real problem of convincingly proving that the system we have developed, *is* actually at that level, and will *remain* at that level for, for example 40 years!

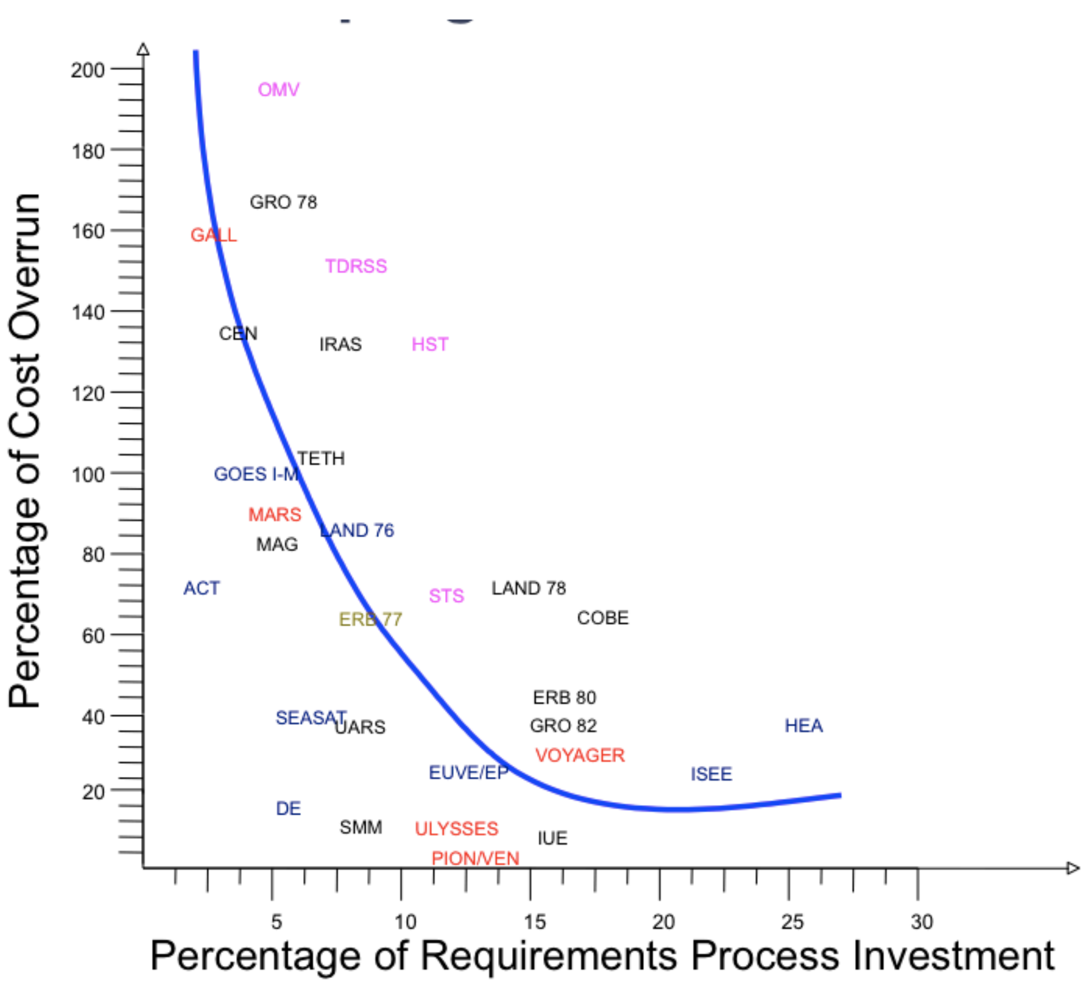
Nobody knows, and nobody can be certain.

One of the problems is that by the time you get historical data, it is likely out of date with respect to the technology you will deploy yourself.

But with no valid and useful historical data, nobody can know.

What we can do is to avoid ‘promising’ specific costs when we really do not know what they are, and what they will become.

Even if we knew a sample of costs for the availability levels, it takes only one single other variable, such as investment in the requirements process, to change the real costs by at least a factor of 2 or 3.



NASA Software project overrun, as a function of investment in requirements. Source: Gruehl (NASA) via Ivy Hooks' book.

Most of you do not have such data. Most of you don't even know about the many factors like this that can influence costs; let alone have relevant historical data about the relationship!

Does begin to feel hopeless? Estimation *is* hopeless! That is what I want you to appreciate. Once you are there, we can turn to solving the problem of controlling costs in a serious way.

3.ARCHITECTURE: If you do your project solutions all at once without learning their costs and interactions – you cannot expect to be able to estimate the results.

You all are well aware that Waterfall or 'Big Bang' projects have a strong tendency to fail [15]. Why?

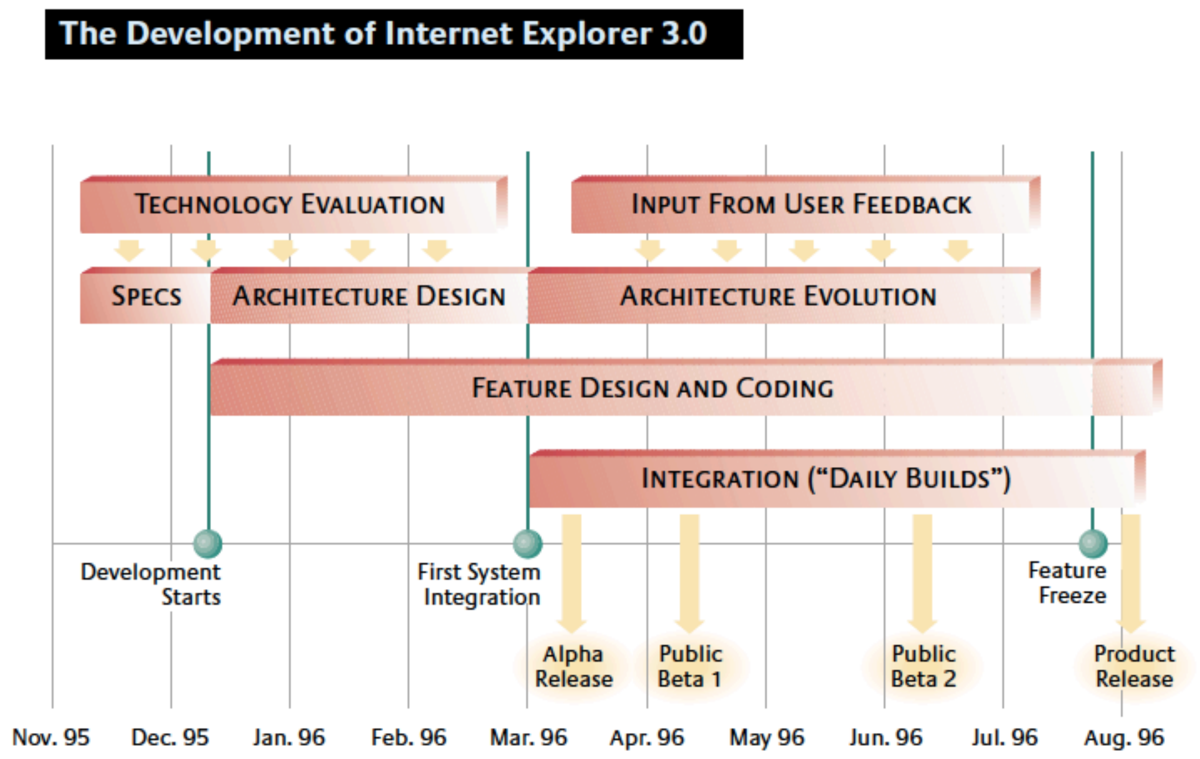


Figure: [Source: 15] An example of a learning and change process.

One of many causes is that they are committed to doing too many things at once. It is therefore difficult to see the effects of any one design, or process, or other factor on the requirements or the costs.

If we implemented critical variables, such as design or development process techniques, one at a time in small increments; then we could see the effects more clearly. We could take steps to change unexpected and bad costs or qualities, before it was too late to change. We could better argue the need for the changes with our own management.

One of our clients, Confirmit, does this very well. They measure the effect of a design, on the 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 leaps in a varied set of quality measures, which help them master their market. [16]

	Design Idea: Step 9 - Recoding			
Requirements	**Estimated Scale Impact**	**Estimated % Impact**	**Actual Scale Impact**	**Actual % 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%**

Figure: Here is a simplified version of the Impact Estimation table [3,2] for Evo Step 9, "Recoding" of the 'Market Recoding' project. 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% of the way to the target level [16].

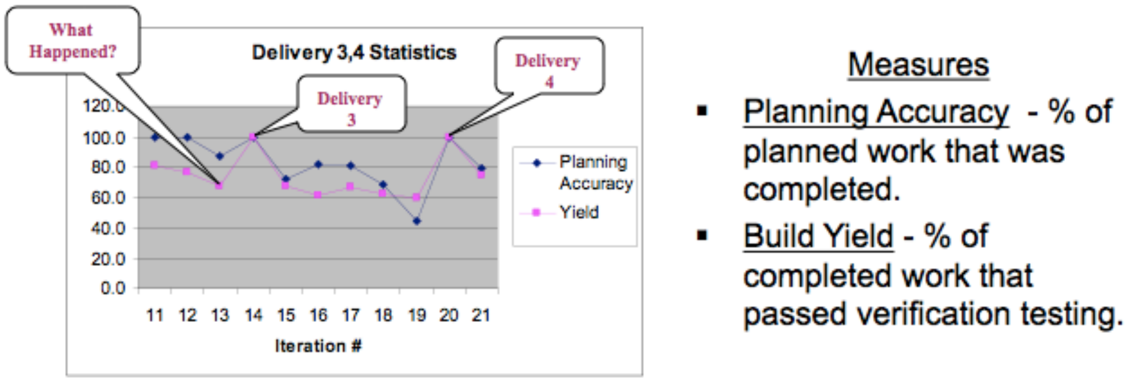
Single cause versus effects, is a fundamental scientific analysis principle. One cause, one effect, keep all else constant - if you want to understand what is going on. Even though reality is that multiple causes combine to give multiple effects.

It requires some discipline to chop things up that way, and build realistically measurable integrated complex systems. But it is the price we have to pay to get some control over costs and qualities. More later.

4.STAFF: If a complex and large professional project staff is an unknown set of people, or changes in mid project – you cannot expect to estimate the costs.

Even if you had a track record for a defined and constant team, you would still have significant problems using that information to understand their productivity, and consequent time and money costs. But we normally cannot be sure *who* will staff our projects, or even if planned staffing can or will be carried out.

This gets even worse with offshore and outsourced projects. Honeywell discovered that top staff had been swapped with less-competent staff, only by monitoring short-term numeric feedback, and finding the root cause for worse performance. [17]



Measures

- Planning Accuracy - % of planned work that was completed.
- Build Yield - % of completed work that passed verification testing.

From Honeywell [17] case where an offshore swap-out of qualified developers was caught by sensing short term changes in the quality of work done.

So, unknowns and unpredictable variations in people, are yet another reason why it is difficult to make project cost predictions.

5. SENSITIVITY: If even the slightest change is made after an 'accurate' estimation, to any requirements, designs or constraints – then the estimate might need to be changed radically. And – you probably will not have the information necessary to do it, nor the insight that you need to do it.

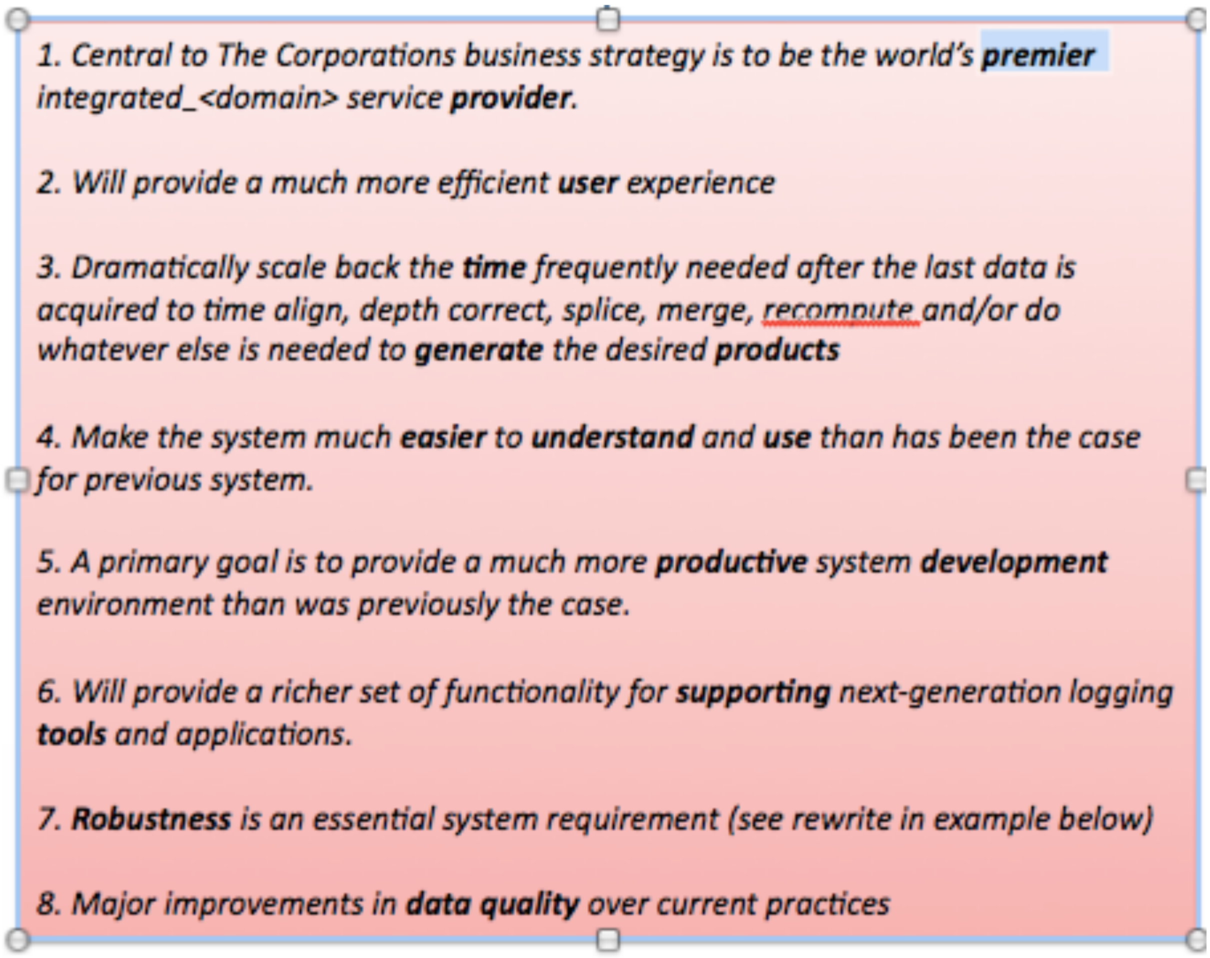
You already have the example above (Principle 1) where 00.08% increase in availability for the AT&T 5ESS software cost 8 years with thousands of developers. We were making the point that system qualities are major cost drivers. But there is an additional point that these cost drivers can be very sensitive to apparently small changes in requirements, or in actually delivered quality.

Costs can be sensitive to other drivers than performance and quality requirements. They can be sensitive to resource constraints – 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 in practice, then the real costs might vary widely as a result. The costs might be surprisingly higher costs initially, or they may come as a 'big surprise', 'due to unforeseen factors' later – giving the true costs.

The unpleasant fact, based on our decades of international practices, 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. [17, 3].

Mm



*1. Central to The Corporations 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 is acquired to time align, depth correct, splice, merge, recompute 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 previous system.*

*5. A primary goal is to provide a much more **productive** system **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 (see rewrite in example below)*

*8. Major improvements in **data quality** over current practices*

Real case study example [17] of the primary objectives for a project that took 10 years before any delivery of these objectives, and cost over $160,000,000. The complete lack or measurable precision in these primary project objectives is, in my view, the primary reason for the time delay and costs. Most managers allow such things to happen on most projects. They lose control of costs right here. Notice that all objectives refer to *qualities (of an existing system). See the reference slides [17] for examples on how to write these more clearly, testably, measurably.*

Conclusion: Risk Principles

The Risk Principles, 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. Any such estimate should be regarded as highly suspicious by managers.

In fact, of the staff doing the estimate are themselves not explicitly aware of this fact, their competence is dubious. If they give fair warning, like this for example:

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

- ***We could use the estimates as a framework budget.***
- ***But we would have to evolve the system in small steps, and learn from experience what the real costs, and the 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 [6, Mills].***

If so, they have given management fair warning, and also a prescription for 'success'. They have understood the Control Principles in the second half of this paper!
Our 'estimation' reality is this:

- Management can decide, based on the projected value of the system, how much they can spend on it, that is the 'budget'
- They can decide based on the market situation, when they want the values to be delivered, that should become your deadlines
- They can then design the project organization to deliver value when they want it, at a cost they find affordable.

- If even the simplest and smallest (weeks, months) attempts to deliver value at satisfactory time and cost fails, then management has an incompetent team or an impossible project, and they should take that as a warning to stop or change.

This is of course a big change in the way most managers manage IT or software projects. Hopefully you are a manager reading this who are daring enough to improve project management, and distinguish yourself!

The beauty of the Control Principles is that it does not take a long time to prove they work in practice. To yourself, your team and your own management.

WHEN ESTIMATES MIGHT WORK: 'I estimate that the cars will stop at the red light'

There are conditions where making estimates *might* work well enough for practical purposes; or might *seem* to work.

• effort on the project ceases, when deadlines and budgets are used up.

• the qualities and performance of the system are not yet at necessary required levels, but people have no particular better expectations, and they are prepared to improve qualities over time to reach satisfactory levels.

• staff/contractors are highly motivated to NOT exceed budgets and deadlines

Stopping, when dubiously estimated resources are finished, does not prove the estimates were ever correct. It just proves that you can stop, and deliver something, without being called a clear failure. Usually you can do *that* with no effort or cost whatsoever, 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.

Ill. 1 Estimated Stop

The Control Principles

Introduction

We have tried to establish that there is no reasonable way to get useful estimates for non-trivial software projects. If your real costs destroy profit margins, or planned return on investments, or destroy your management reputation – they are not useful. Our estimating capability usually fails this simple test. But we have also learned many ways of covering this truth up [9,10].

I am sorry if I have yet failed to convince you of this. Come back to this paper when your own experience finally convinces you this is true (I have 52 years experience). Assuming you are still interested in what to do about the estimation problem, I will offer the 5 Control principles below.

We can simplify them as follows:

1. **do something of value in a short time**
2. **measure values and costs**
3. **Adjust what you do next, if necessary**

Repeat until you no longer can find value for money.

Advantages with this method

1. you can*not* waste much time or money before you realize that you have false ideas
2. you *can* deliver value early, and keep people happy
3. you are forced to think about the *whole* system, including *people* (not just code)
4. so you are destined to see the true costs of delivering value – not just code costs.
5. you will learn a general method that you can apply for the rest of your successful career

Disadvantages with this method

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

So here are the Control Principles in detail

6. LEARN SMALL: Do projects in small increments of delivering requirements – so you can measure results and costs against estimates.

All software projects can be broken into early and small increments. Not merely increments of *building*, but more importantly – increments of *delivering value* to your *stakeholders*.

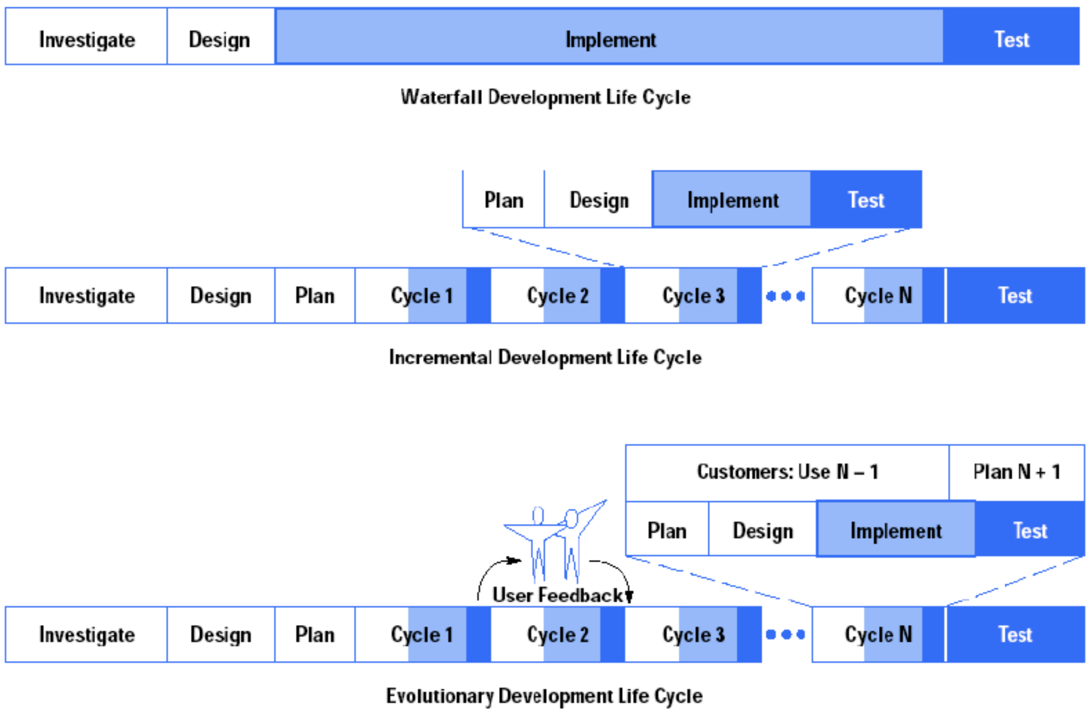


Figure 1. Life Cycle Concepts (Cotton 1996)

III. Concepts of small delivery cycles with stakeholder feedback. From HP, a client who uses our Evo method. [18, 14, 15]

I have experienced that almost all projects can, if they really want to, start delivering some value to some stakeholder, next week and every week thereafter. But most projects do not even try, and don't even try to learn how.
Leaving aside how to do so for a moment [18, 19] – if you can do this then you get control over value for money.

- You can select the highest available deliverable value, each step
- And deliver it to the most critical stakeholder
- And prove that your designs actually work
- And get some sense of the time, and money costs

You get a stream of high value, prove that it is really value, and have full control of the costs step by step. The steps are usually one week at a time, in my practice. Assuming you can deliver reasonable value for effort spend week after week – a surprising thing happens:

- People cease to care about 'the deadline'
- People cease to ask for estimates of the monetary budget
- You are strongly encouraged to keep on going, until value is less than costs
- You end up delivering far more real value than other projects do, well before 'the deadline' (that would have been set, and would have been overrun)
- Management shifts focus from budget, and costs - to Return on Investment

I sometimes simplify this method by calling it the 1.1.1.1.1.1.1 method, or we could call it the 'Unity' Method

- Plan, in 1 week
- To deliver at least 1%
- of at least 1 requirement
- To at least 1 real stakeholder
- Using at least 1 design idea,
- On at least 1 function of the system

It is amazing the practical power of this simple idea of Unity

If you really try, and management persists in encouragement and support, it almost always works.

As one outside-the-box example I succeeded in helping 25 aircraft projects at Boeing (El Segundo, CA then MD) to do this in practice and get approved for the small steps. No exceptions. It is not a principle limited to software, though most of my experience is there.

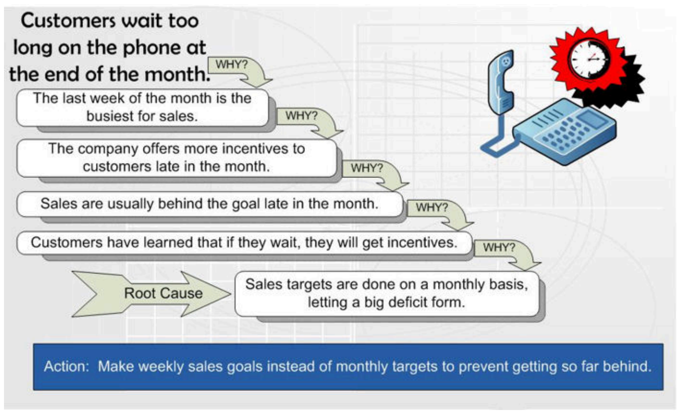
7. LEARN ROOT: If incremental costs for a given requirement deviate negatively from estimates – analyze the root cause, and change anything about next increments that you believe might get you back on track.

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

The we need to measure both the impact, to make sure it is roughly as estimated, and note the costs. If we have to continue tuning the design to reach the required quality levels, then the taximeter is still ticking. Keep track of those real costs, and note the cost of getting to the required level.

If you get some surprises about the real costs, and you will get them – then we have to learn as much as possible as early as possible from the experience.

One process for learning, is to seek the root cause [22] of the deviation from our estimates. If we can find the root cause, and therefore try something to avoid it, we can reduce costs and improve quality levels towards our estimates. A simple strategy for root cause thinking, that I have been very happy with in practice is asking 'Why?'. The Japanese 'Five Whys' method (http://en.wikipedia.org/wiki/5_Whys) hints at the need for iteration until you hit a level you can deal with effectively. I have found that people very frequently are at the wrong level of understanding a problem.


Customers wait too long on the phone at the end of the month.
WHY?
The last week of the month is the busiest 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 will get incentives.
WHY?
Root Cause
Sales targets are done on a monthly basis, letting a big deficit form.
Action: Make weekly sales goals instead of monthly targets to prevent getting so far behind.

Ill. Simple example of applying 5 Whys

David Long, wrote me referring to a very large software project (AT&T Switching 5ESS) that:
"But, from what I saw, the greatest improvements came from a small amount of work that was done as the result of specific root cause analysis performed on specific outages." [21]

You might conclude that there is a lot of cost reduction leverage to be had from root cause analysis. Solving the critical few problems that are frequent and damaging.

But we have to ask, how can you factor such processes into an initial cost calculation? You cannot. You have to do it as you work, and maybe reduce costs to the level some optimist 'estimated'.

8. PRIORITIZE CRITICAL: You will have to prioritize your most critical requirements and constraints: there is no guarantee you can achieve them all. Deliver high-value for resources-used *first*.

A cost estimate, once it becomes a deadline or a budget, is a 'limited resource'. One of the smartest ways I know to deal with limited resources is intelligent prioritization. [23]

Instead of implementing all your designs at once, and hoping to meet requirements and cost estimates – we deliver a little bit at a time – and see how things work and what they cost.
By a little bit at a time I mean something like evolutionary delivery cycles of a week, or something like 2% of the overall project cost scope.

Using an impact estimation table we pick the designs that give the estimated best value for their costs. We implement them in such a way that we can get acceptable feedback on costs and value delivery (in terms of meeting requirements).

With a bit of luck you will both deliver interesting value to stakeholders very early, and you will learn what works and what doesn't in practice.

If you do hit a deadline or run out of budget (possibly because of a budget cut, or deadline being moved up earlier!), then you will at all times have a complete live real system, with high value long since delivered.

In such situations, the whole concept of the estimates, the deadlines and the budgets is not so important any more.

In the Cleanroom Method, developed by IBM's Harlan Mills [6] they reported:

•*"Software Engineering began to emerge in FSD" (IBM Federal Systems Division, from 1996 a part of Lockheed Martin Marietta) "some ten 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 [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 ten years, FSD has managed some 7,000 person-years of software development, developing and integrating over a hundred 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

[IBM Systems Journal No. 4, 1980, p. 415], Reprinted IBM SJ Vol. 38 1999, pp 289-295.

Nobody does it better! Harlan Mills told me that they had to solve the persistent problem of cost overruns, in relation to the contracted estimates, because the government was getting smart 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, they could just as well leave the business.

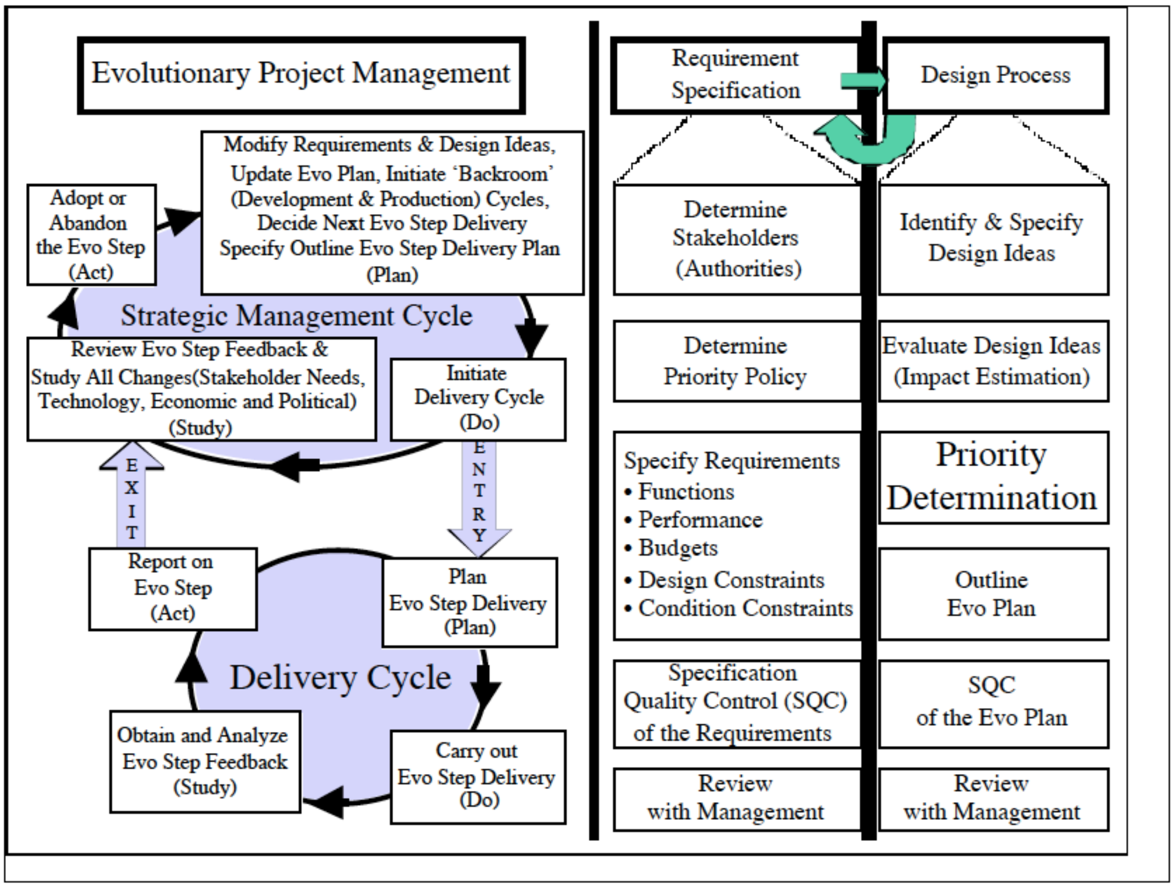
Notice the '45 deliveries'. That is 2% delivery cycles. They were using intelligent feedback and learning. They actually had very sophisticated estimation technology based on a thorough collection of experiences (Walston and Felix IBM SJ 1/1977). But this did not give them the accuracy they needed to avoid losing money. Lets us say they had a 40% profit margin, and they could be wrong by 40% to 200% (The NASA range, see above diagram). They would still lose money on most contracts.

So, they had to compensate for their estimation inaccuracy by incremental feedback and necessary change to come in 'on time and under budget every time'. Mills Colleague Quinnan describes their process of intelligent dynamic prioritization in detail (IBM SJ 4/1980).

A thorough explanation of the processes I recommend (diagram below) to do intelligent prioritization is found in my Competitive Engineering book [3]. Nothing less will suffice for large and complex systems (hundreds of staff, years effort, $100 million or more in cost). But for smaller systems (small teams over a few months) my clients have made good simple practical adaptations [14, 16] of the essential ideas.

My 'Agile' friends have finally adopted small iterative cycles [5], but they have failed to adopt the notion of measurement of value and quality – which is essential for larger projects, and some smaller ones. So the value prioritization is crude and too subjective.

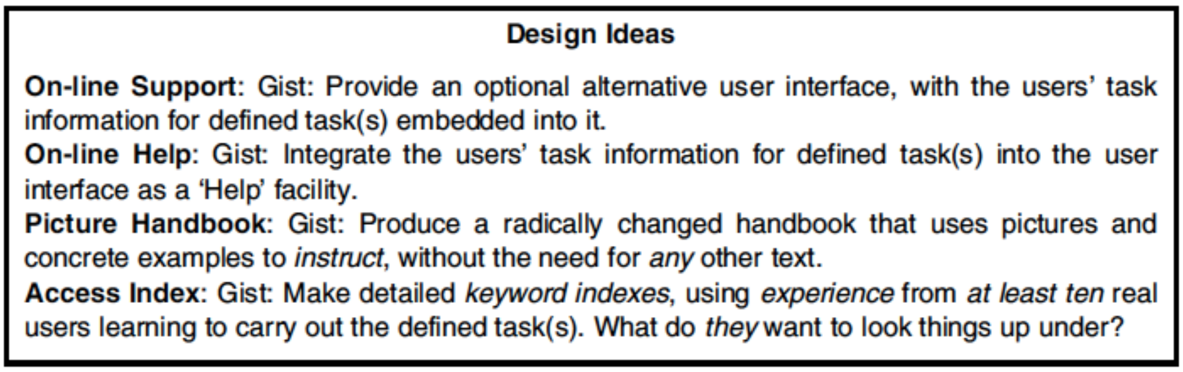
Anyway – here is my model of the processes surrounding intelligent iterative prioritization.


Evolutionary Project Management
Modify Requirements & Design Ideas, Update Evo Plan, Initiate 'Backroom' (Development & Production) Cycles, Decide Next Evo Step Delivery Specify Outline Evo Step Delivery Plan (Plan)
Adopt or Abandon the Evo Step (Act)
Strategic Management Cycle
Review Evo Step Feedback & Study All Changes(Stakeholder Needs, Technology, Economic and Political) (Study)
Initiate Delivery Cycle (Do)
EXIT
ENTRY
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 & 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

Priority management as an iterative process: The left-hand of the diagram shows the Strategic Management Cycle and the Delivery Cycle within Evolutionary Project Management (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. Note Impact Estimation and Priority Determination are within the Design Process [23a].

9. RISK FAST: You should probably implement the "highest value with regard to cost and risk" design-ideas early.

We can use Impact Estimation [3] tactics to identify the value with regard to cost and risk items in advance, so we can try them out first.



Design Ideas

On-line Support: Gist: Provide an optional alternative user interface, with the users' task information for defined task(s) embedded into it.
On-line 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 ten* real users learning to carry out the defined task(s). What do *they* want to look things up under?

Source: Competitive Engineering book [3] page 267, Chapter on Impact Estimation.

If we look at the Design Ideas (from real case at Ericsson) above, defined only at the 'Gist' level (there was more detail to the definition, but this is OK for our purposes here). Which of the designs is 'high risk' ? Well there is no documentation as to what our experience would lead us to expect of costs or results. So there is no difference. Unfortunately, too many design specifications are at this level. A few descriptive words and we are done. No estimates for costs, no estimates for expected impacts on requirements. And no estimates of the certainty of the estimates.

I have invented a method for bringing out such information, so that we can understand what is risky, and act accordingly (trying out the high risk stuff early – this principle).

Below is a real example, we did for our client.
The main requirement we are focussing on is called 'Learning', and the numerically specified requirement was to reduce the learning time from 60 minutes to ten minutes.

Scale Impact: the first set of estimates are our best guess as to what the result would be if we implemented each design. On-line Help is *estimated* to get us to the required 10 minutes.

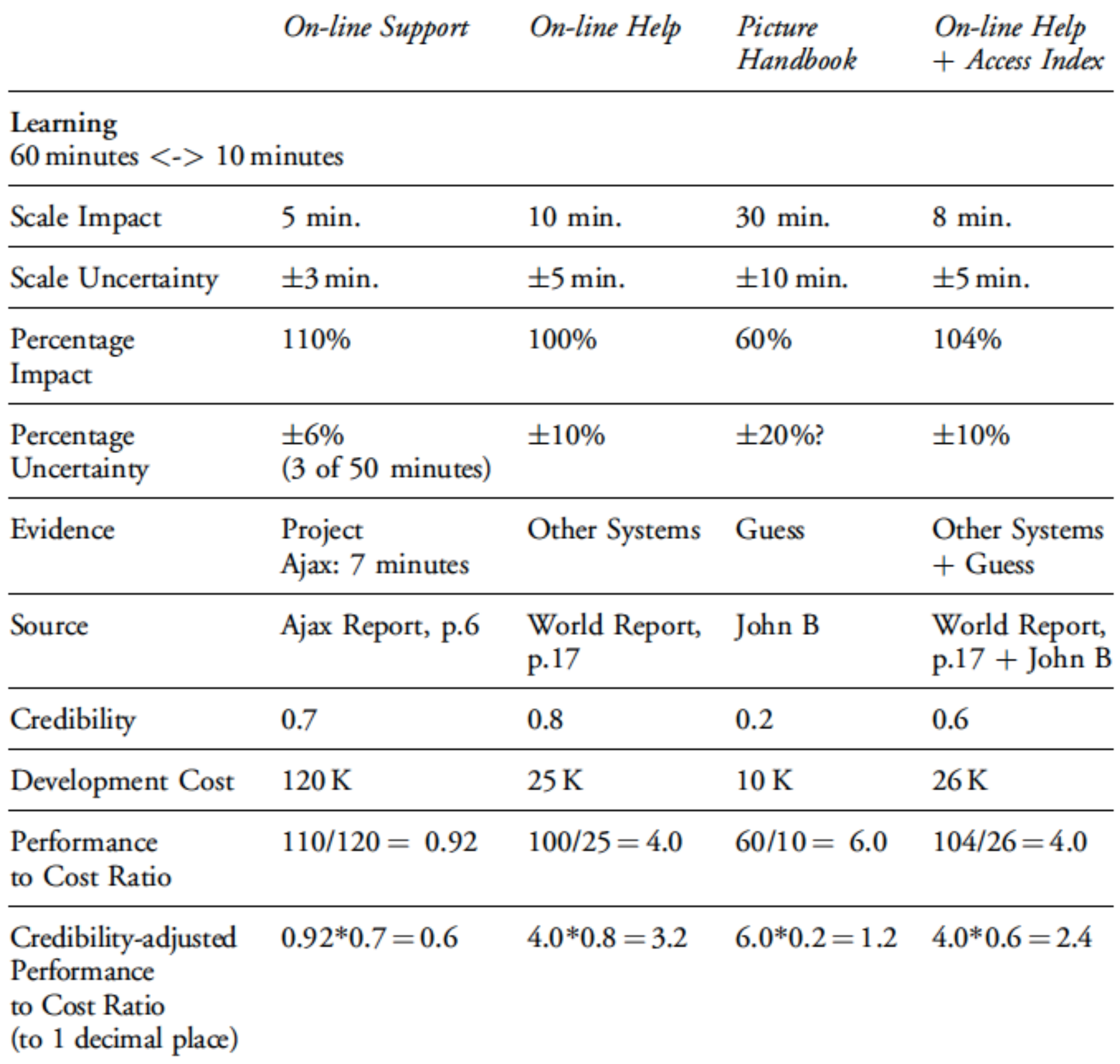
Scale Uncertainty: but, to be honest we are unsure of all of the estimates and have given another estimate of the uncertainty. The best case and worst case borders. One expression of the risk that the real result will NOT be our initial estimate. In this case the ± 5 minutes for On-line Help mean that we think the worst we could get is a result of 15 minutes (10+5). And the best is 5 minutes (10-5).

The % Impact is just a way of expressing these same estimates in direct relation to the Learning requirement level (10 minutes = 100% of the objective). This is not important for our purposes here. We use the % Impact to make comparisons to *other* scales of measure for *other* requirements that we might well be looking at simultaneously to make a risk decision. But not in this simplified example!

Now based on this (Scale Impact and Uncertainty), which one of the 4 designs is the smartest one to try out first? On-line Help looks good to me, and Picture Handbook looks like a loser. If On-line Help succeeds, I would not need the other design ideas. So our principle does not mean to recommend that we do the Picture Handbook design (high risk of NOT giving us the result we want at all). We are primarily interested in the risk it will cost us more than we estimate, or can tolerate, to get the result. So we have to take a look at some other information.

How good is the estimate of impact? Who made it, on what basis? It is rare to find designers for software systems documenting the who and why of any estimate, let alone the estimate of impact on a numeric quality requirement, like Learning.

Evidence/Source: We document the 'Evidence' (on what basis) and a 'Source' (person or document for the Evidence, 'who'), as best we can. And based on that, we assign a Credibility score (0.0 worthless, 1.0 perfect, based on a table [3]. In this case On-Line Help came out best (0.8) so we can stick with our belief that it will give us the best result. But, that does not settle the question of the costs and their uncertainty!



	On-line Support	*On-line Help*	*Picture Handbook*	*On-line 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	120 K	25 K	10 K	26 K
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

Source: Competitive Engineering book [3] page 267, Chapter on Impact Estimation.

Development Cost: now we can make an estimate of the development cost (and/or any costs aspects that interest us, such as development duration). On-line Help is estimated to cost 25,000.

Credibility-adjusted Performance to Cost Ratio: when we modify the costs with the 'Credibility' factor – a rough measure of how well we know the Design Impacts and Costs based on history (Evidence and Source), it becomes even more convincing that we should stick with On-line Help as the design to try first. (3.2 being the largest value for money, credibility adjusted.). We have one last trick up our sleeve to make a decision, though we considered it above. We could make a ± estimate for the costs, and consider how wide (risky) it is, and what our worst case (estimate plus the highest cost border).

Do you feel lucky today punk?

This might all seem like a lot of 'bureaucracy' to some of you. But it is a lot cheaper than just diving in, and implementing unevaluated designs, and failing, as software people so often seem to prefer. It is in fact a reasonable fact-based reasoning or logical process that you should try to do, at least when it *is* a lot cheaper than leaping before you look.

My assertion is this, if you consistently choose the safest bets, the iterative implementation options that promise, based on good evidence, to deliver best requirement-value for costs, then your costs will be under better control.

But you have to do this in an iterative sequence. You cannot play all your bets at once. You have to be prepared for learning to estimate better, learning to provide better sources and better evidence. You have to 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 you need to implement – you should do the potentially high-value to cost, but 'high risk' designs early.

This assumes that there is high value *estimated*, at low cost *estimated*, but that the ± uncertainty is very wide. Example 80%±70% for a quality or a cost budget.

In that case there is an argument that we should find out whether the design is as promising as we optimistically would like it to be, or not.

There is a high value in knowing the reality here.

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 to be able to retreat, should it be falsely overoptimistic!

10. APPLY NOW: Learn early, learn often, learn well, and apply the learning to your current project.

The tenth principle is implied and discussed in connection with the other principles. If you want to get control over costs and budgets, then you have to get the truth of the cost of meeting your requirements.

You have to humbly recognize that neither you nor anyone else is certain of the costs. Would you bet your life that you are right in your cost estimates? Would you deduct the amount you are in fact wrong from your pay? Of course not. So, you should not spend your employers money unnecessarily because of your own arrogance and incompetence – and lack of ethics.

You need to start learning the true costs of your designs for meeting your requirements as early as possible. I prefer to start learning for real the second week of any project. Meetings and opinions just can't beat reality. Then I believe the pace of learning cycles should be weekly until all requirements are delivered, or until you run out of resources.

I won't give you a hard time if you believe in 2 weeks or 4 weeks cycles, but I would argue that you have something to learn about decomposing projects [18]. Remember the 1.1.1.1.1.1. principle of decomposition discussed above? If you use a month to measure cost and effect, then you risk losing 3 weeks of time and 4x as much resource to find out that you do not know what you are doing yet!

Summary

The world of software is complex enough in itself – some say the most complex world we humans make. But software is always part of a larger system of machines, people, laws, markets, politics. So understanding the *real costs* for developing initial solutions, to deliver competitive levels of performance and quality, is near impossible up front.

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.

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 un-measurable and unpredictable future. Something more than the suggested principles apply to this problem. We are talking about how to engineer long term cost drivers into the system, initially and gradually [25].

Management needs to base their planning on the cost they are prepared to pay to deliver specified values (as specified in quantified requirements). This is keeping your eye on value of money, or profitability. Let us call this the *maximum profitable cost (MPC).*

Management needs to also decide the time value of delivering specific system value levels (such as 99.99% availability within 12 months). Let us call this the *high value delivery timing (HVDT).*

Management then needs to make sure that their projects are done one step at a time (2% steps of the MPC and/or the HVDT). Management needs to make sure that projects are organized to deliver highest possible value as early as possible, and that projects learn what is real very quickly.

REFERENCES

[1] Peter J. Morris, The management of projects, Telford, London, ca 1995

- Morris concluded that successful management of large projects demanded some form of evolution feedback and learning.
- http://www.indeco.co.uk
-
- [2] Gilb, Tom, Principles of Software Engineering management (chapter on Estimation), Addison-Wesley, 1988. http://www.amazon.com/Principles-Software-Engineering-Management-Gilb/dp/0201192462
-

[3] Gilb, Tom. Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering, Using Planguage, 2005, Elseveir. ISBN 0 7506 6507 6.

- http://www.amazon.com/dp/0750665076/ref=rdr_ext_sb_ti_sims_2
-

[4] Boehm, Barry. CoComo and CoComo II

- http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html
- http://www.amazon.com/Software-Cost-Estimation-Cocomo-II/dp/0130266922

[5] Larman and Basili: History of IID methods, IEEE Computer June 2003

- http://www2.umassd.edu/SWPI/xp/articles/r6047.pdf
- http://www.highproductivity.org/r6047.pdf
- Specific mention of Gilb's contribution.

[6] Mills, Harlan IBM System Journal NO. 4 1980

- http://portal.acm.org/citation.cfm?id=1662229.1662231
- http://portal.acm.org/toc.cfm?id=1662229&coll=GUIDE&dl=GUIDE&type=issue&CFID=97759667&CFTOKEN=76112285
- Mills, H. D. 1980. The management of software engineering: part i: principles of software engineering. *IBM Syst. J.* 19, 4 (Dec. 1980), 414-420. DOI= http://dx.doi.org/10.1147/sj.194.0414

[7] http://en.wikipedia.org/wiki/Estimation

- "In mathematics, *approximation* or *estimation* typically means finding upper or lower bounds of a quantity that cannot readily be computed precisely. While initial results may be unusably uncertain, recursive input from output, can purify results to be approximately *accurate, certain, complete and noise-free."*

[8] Gilb, 'No Cure No Pay' paper (38) and slides (85):

- http://www.gilb.com/tiki-download_file.php?fileId=38
- http://www.gilb.com/tiki-download_file.php?fileId=85

[9] **Collins97: Tony Collins, with David Bicknell, CRASH: Learning from the World's Worst Computer DISASTERS,**

- Paperback, 1998, Simon and Schuster Ltd., £9.99, 428pp, ISBN 0-684-81687-3
- *An incredible must-read book about poor management decision-making in the IT sector in UK (mainly) and the US*
- *http://www.amazon.co.uk/CRASH-Learning-Worlds-Computer-Disasters/dp/0684816873*
-

*[10] **David Craig and Richard Brooks***

- *Plundering the Public Sector: How New Labour are letting consultants run off with £70 billion of our money*
- *http://www.amazon.co.uk/Plundering-Public-Sector-David-Craig/dp/1845293746*

[11] The Lucent AT&T 5ESS Case.

- *A Case study in 1980's in CACM said that the 4ESS Switching system had availability of 99.90% and the new 5ESS targeted and achieved 99.98% availability. The Case study mentioned this took 8 years. Other sources told me that the project had 2,000 to 3,000 participants, depending on time you looked at it.*
- *Possible source: [11b*
    - *http://www.mdavidlong.com/resume_m_david_long.pdf*
    - *"5ESS system reliability of 99.9999%"*

*[12] Magne Jørgensen, [12a] **Practical Guidelines for Expert-Judgment-Based Software Effort Estimation. IEEE Software May/June 2005.***
[12b] See also his
http://simula.no/research/se/publications/Simula.SE.375
Slides 2008," Software Development Effort Estimation:
***Why it fails and how to improve it**." Notice that this excellent analysis of estimation problems is focused on how to improve initial estimations. It does not discuss the main theme of this paper, that we have to iteratively adjust the project to the desired costs. His remark "Has the project planned regularly re-estimations and re-risk analyses to identify effort overruns as soon as possible?" does not cover the tactic of redesign of product and process to get control of costs.*

[13] Kjetil Moløkken-Østvold and Magne Jørgensen**: A Comparison Of Software Project Overruns—Flexible Versus Sequential Development Models**, IEEE

Transactions On Software Engineering, Vol. 31, No. 9, September 2005*. "The results support the claim that*
projects which employ a flexible development model experience less effort overruns than do those which employ a sequential model."
http://simula.no/people/kjetilmo/?searchterm=papers

[14] Sharma Upadhyayula, " RAPID AND FLEXIBLE PRODUCT DEVELOPMENT: AN ANALYSIS OF SOFTWARE PROJECTS AT HEWLETT PACKARD AND AGILENT", Massachusetts Institute of Technology January 2001. http://www.gilb.com/tiki-download_file.php?fileId=65
This study is based on the use of Gilb's Evo method as adopted by HP.It has considerable data of factors related to cost.

[15] ***Alan MacCormack,*** *Product-Development Practices That Work:*
How Internet Companies Build Software, WINTER 2001 MIT SLOAN MANAGEMENT REVIEW*, page 75 on. "Successful development was evolutionary in nature".*
http://hbswk.hbs.edu/item/2201.html (extract)

[16] Confirmit case: a Norwegian survey software and service company. Using our Evo method excellently since 2003.

- http://www.gilb.com/tiki-download_file.php?fileId=32
-

[17] Jerry Berntsen, A Case Study in Globally-Distributed Lean Software Development. ICSPI 2007 Conference.

- http://www.icspi.com/tracks/Jerry_Bernsten_Lean_Software_Development.htm

[17] On the subject of quality and completeness of requirements that might determine costs:

- http://www.gilb.com/tiki-download_file.php?fileId=180
- Slides and cases of top level critical requirements

[18] Gilb, Decomposition of Projects: How to Design Small Incremental Steps INCOSE 2008

- http://www.gilb.com/tiki-download_file.php?fileId=41
-

[19] Value Planning Slides for Scrum Product Owners

- http://www.gilb.com/tiki-download_file.php?fileId=353

[20] Case Study: Mgt Objectives, QC and a little about 25 Evo projects for aircraft engineering.

- http://www.gilb.com/tiki-download_file.php?fileId=254
-

[21] David Long data. Personal email to author 21 July 2010. "M. David Long" dave@mdavidlong.com.

- "I do believe your numbers are certainly in the right ballpark though (if you count all $'s and effort spent on quality improvement) between the 1AESS and 5ESS."
- " We implemented design reviews and code inspections (based mostly on your work in the '80s"
- "When you're moving from 4 "9's" to 5 "9's", you have to work on the outliers with specific technical solutions... not just process quality improvements. "
- "But, from what I saw, the greatest improvements came from a small amount of work that was done as the result of specific root cause analysis performed on specific outages."
- 'There was much money and many years spent within the 5ESS organization in the name of quality (as measured by availability) improvement. '
- "In 2000-2001 we achieved "6 9's" of availability."
- 'PS After leaving Lucent in 2001, I went to a start-up (Taqua) where we were able to replace most of the functions of the 5ESS (starting from scratch) in 1/100th the space (physically) in a total of 350 staff years of effort, operating at 5 "9's" of availability. '

[22]. Eliyahu M. Goldratt, The Choice, North River Press, 2008. This discusses root cause analysis, among other subjects. http://www.northriverpress.com/product50.html

[23]. Gilb Prioritization Papers: http://www.gilb.com/tiki-download_file.php?fileId=60
"Managing Priority" (23a)
http://www.gilb.com/tiki-download_file.php?fileId=48
"Choice and Priority" (23b)
These papers explain my dynamic prioritization ideas: same ones nature uses to operate your body.

[24]. http://www.gilb.com/tiki-download_file.php?fileId=138
Gilb: **Designing Maintainability in Software Engineering: a Quantified Approach.**

This paper details how to specify system qualities related to long term operational costs of the system (Adaptability). This would be a starting point for architecting control over long term operational costs of the system.

[25] http://simula.no/research/se/publications/Grimstad.2006.1/simula_pdf_file

Software Effort Estimation Terminology:

The Tower of Babel

Stein Grimstad, Magne Jørgensen, Kjetil Moløkken-Østvold

Simula Research Laboratory

{steingr,magnej,kjetilmo}@simula.no