

Using Lean LSS to Improve Software Implementation Efforts

By Gary A. Gack

This article will consider two scenarios that are unfortunately quite common in software implementations:

Scenario 1

- “We do an enhancement prioritization process with our customers at our annual user group conference, but somehow it just doesn’t seem to work very well. We end up with a bunch of stuff that doesn’t seem to have any sort of overall theme – almost features without a rationale. We need another way to work this!”

Anchoring Requirements in Business Outcomes

One of the common misconceptions concerning Lean Six Sigma (LSS) is that it’s all about statistics – in reality it’s much more than that. LSS involves disciplined use of *language* data as well as numbers, especially when applying Design for Six Sigma (DFSS). Requirements, after all, mostly get described in language, and typically not very precise language at that. We often, as in this case example, end up with a laundry list of features and functions whose coherence and central themes are often very unclear, even to those who may have given us the requirements. We’ve all had the experience of getting half way through a project and realizing that both the development team and the customer are wondering “now why is it we’re doing this particular feature???”

When LSS thinking is applied to requirements the focus is often quite different than the typical “what are your requirements?” approach. Instead the focus is on understanding the customer’s “Critical to Quality” business objectives – developing a rich understand of what the customer is trying to accomplish and how value will actually be generated. At first glance this may seem a fine distinction, but in practice it leads to a very different mindset that creates very different outcomes. Implications of this different mindset include:

- **Desired business outcomes precede features and functions.** A DFSS approach will focus first on the business results, described in financial terms, that are the reason a system is being developed or enhanced. Certainly most projects begin with some sort of high-level statement of business objectives that justify initiation of the project, but that focus is often lost when the team starts to “find out what they want” – by the time the project is a couple of months old few remember the initial rationale. Most projects quickly lose sight of the “why are we doing this?” and “how does this feature/function contribute to realization of the expected business value?” point of view – losing the connection between “what they asked for” and how satisfying those wishes will produce business value.

An accounts receivable system, to offer a simple example, fundamentally has only one reason to exist – i.e., to facilitate collection of money owed the organization. Even in such a simple example systems are very often built that have dozens of different ways to enter transactions or view amounts outstanding, reflecting the individual preferences of collections and accounting personnel in the various divisions and regions of the organization. Perhaps many of these units were acquired over a period of time. Perhaps they all used different systems and different business processes – some used Oracle, some SAP, some had QuickBooks. They all want to have their reports and screens exactly the way they are accustomed to seeing them – and as a consequence the implementation team builds far more software than is fundamentally necessary, creates many versions of the training, provides help desk support for all the variants. A very large part of this extra effort has essentially no actual business value.

- **Impact based selection of functionality to be delivered.** Instead of "popularity contest" that relies on some sort of voting scheme and/or on the political or financial clout of certain stakeholders, a more fact-based approach is typical when DFSS is being used. A scorecard based on an adaptation of the Pugh method appropriate to the circumstances can provide a formal mechanism that facilitates objective evaluation. Proposed features and functions can be rated against an agreed set of CTQ attributes that reflect not only the business outcomes but also important non-functional attributes of a solution that meets all "well-founded" customer requirements (as distinguished from wishes and matters of taste or style). Attributes that may be rated for each proposed feature/function might include some of the following:
 - The contribution it makes to financially measured outcomes – if we add this feature will our collections improve?
 - The contribution it makes to the cost of operating the system – if we add this feature will it reduce our operating cost or cost of ownership?
 - The contribution it makes the efficiency of the personnel using the system – if we add this feature will it reduce the time it takes to enter transactions? The time it takes to perform a collection activity?
 - The contribution it makes to deployment of the system – if we add this feature will it reduce training time? Reduce development of training materials?
 - The contribution it makes to system reliability – does it make the system more foolproof? Is the cost of the feature consistent with the associated failure risk?
 - The contribution it makes to security – does the feature make the system less vulnerable? Is the cost of the feature consistent with the associated risk?
 - What portion of system users will use the feature – is the user base impacted consistent with the development costs?

Questions such as these (and certainly there could be many others relevant to a particular situation) can be an effective screen that prevents gumming up the works with a lot of low-value stuff. When ratings have been assigned to proposed features and costs have been estimated for each it is a relatively simple matter to use the resulting scores as the basis for decisions on what to include in light of the available budget.

A software firm I know applied this approach to a major new release – when they presented their approach and results they received a standing ovation from the user group members who participated in the identification of potential features/functions and in the ratings process. They saw a significant increase in upgrade revenues, and for the first time in years the internal friction between development and marketing was reduced to a low boil.

Is this rocket science? Of course not! Did we need advanced statistics? No way! What was needed, and what DFSS supplied, was a disciplined, fact-based process that was visible, understandable, and defensible. Certainly there was room for argument on the ratings, and many arguments occurred, but in the end, everyone involved understood how and why the decisions were reached. Internal and external alignment was better than it had been in years.

Scenario 2

- “Every time we seem to go through the same drill with schedule and budget commitments. Our customers insist, understandably, that we provide a preliminary estimate of project costs and schedule very early in the process before anybody really knows what the requirements and scope actually are. We give them our best guess, always making sure we tell them it's a guess. Once we have a signed-off statement of user requirements we go through a careful planning process and come up with a much more realistic estimate of effort and schedule. Our manager takes that to the customer and inevitably comes back and tells us we have to stick with the original guess! Of course we never make it and we're made out to be the bad guys! We're all really fed up with this – what can we do about it?”

Anchoring Schedules in the Real World

Ouch! Everybody who's been doing software projects for even a short while has probably experienced this.

We all realize estimating interacts with requirements issues, but for the sake of clarity we will treat the estimating problem in isolation. Software cost estimating is fraught with all sorts of difficulties, but under the complexity there are some universal truths that are worth understanding. Simplified analogies, so long as they are faithful to reality, can sometimes help us manage customer and management expectations more realistically than might otherwise be possible.

One analogy familiar to everyone, and yet faithful to the real dynamics of software projects, relates to costs and risks associated with vehicle transportation. In these days of high fuel prices, some of us might be thinking more carefully than in the past about the pros and cons of taking the family on a driving vacation in that new SUV.

Perhaps the spouse and kids have always wanted to see the Grand Canyon – a pretty long trip from our home in Florida – about 2500 miles each way. The boss has given us 2 weeks vacation, so we've got to work within that time frame – similar to a fixed deadline for a software project. Also like a software project (sometimes) the essential requirements are not negotiable – spouse and all three kids are coming along – leaving somebody home is not an option. We do have a bit of flex about what we take with us, so to some small degree we can control the weight of our vehicle.

We need to make a pretty good estimate of what this trip is likely to cost – braces for Susie and tuition for Chelsea are putting a dent in our available funds. So, we need to consider our time limit and the expenses involved. Like software projects, these factors are interdependent – i.e., what it will cost (fuel, maintenance, etc) is in part a function of how fast we drive and how much stuff (scope of the requirements) we bring along. And how fast we drive impacts how much time we'll be able to spend actually vacationing (reaping the benefits) and how much time we'll spend driving in the car. Hmmm .. this is getting a little difficult to envision without writing it all down to clarify what our options are and calculating the consequences of the various options available.

We might (if we're really this anal) prepare an analysis something like the following, using 5,000 miles round trip and fuel at \$3.00 per gallon:

Speed (MPH)	Weight (Rqmts Scope)	Drive Time (Duration)	Miles/Gallon (Productivity)	Fuel Cost (Effort)	Risks
50	Nominal	100 hours	18	\$832	Nominal
60	Nominal	83	16	\$938	+ (accidents)
70	Nominal	71	14	\$1,071	++ accidents + tickets
80	Nominal	62	12	\$1,250	+++ accidents +++ tickets

We could elaborate this with more options related to weight and perhaps introduce other factors, but this is enough to get the idea. As with software cost estimating it is possible to construct *scenarios* that illustrate the consequences of choices open to us. Are we willing to risk tickets and accidents (delivered defects and missed schedules and budgets) in order to

have some sort of chance to save 30% of the drive time (duration)? Realizing it will cost 50% more and that we will have to take into account the costs and delays of the tickets and repairs?

Software projects *always* experience exactly these fundamental dynamics, but the actual underlying models that mathematically describe these dynamics are much more complex and non-intuitive than the simple model we presented here. Yet the analogy is perfectly applicable and valid – faster costs more (a *lot* more) and carries greater risks. The increases in risks and costs are dramatically non-linear (exponential) and hence have far more serious and costly consequences than is the case with our simple trip model. In reality it is often true that a 30% compression of a "least cost" schedule (50 miles per hour) will lead to 3x estimated cost and 5-8x delivered defects.

The graphical representation below shows one published example of the sort of model and dynamics that are actually used for software projects. Quant jocks among you may wish to take a look at Mike Ross's article in a recent issue of the ITMPI newsletter for a deeper look at realistic models and formulas of software cost/schedule/defect dynamics. As mentioned above all of the proven software cost models embody non-linear relationships that are virtually impossible to visualize – models are essential to fact-based discussion. NOTE: you don't need to understand the formulas to use the "black box" tools in which they are embodied! See <http://www.compaid.com/caiinternet/ezine/ross-25yrs.pdf> (the source of the diagram below)

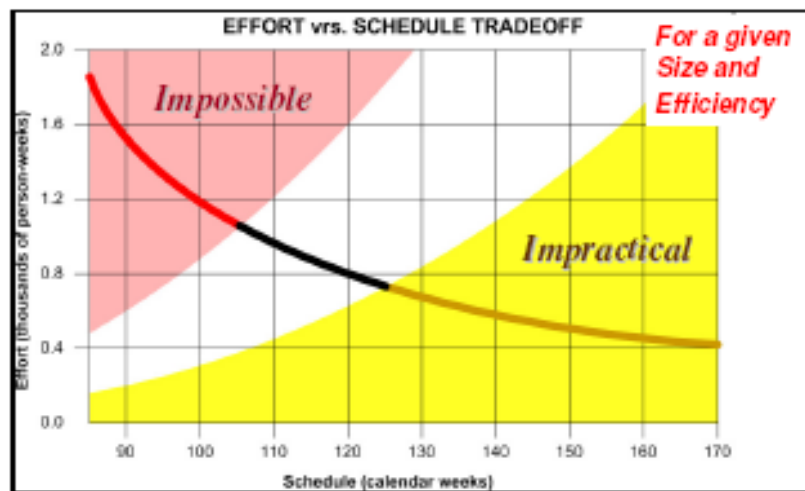


Figure 10. Minimum Effort Limit

The take away from all of this is that use of LSS leads to a fact-based culture in which key players become quantitatively literate – they become comfortable with the idea of modeling outcomes and making decisions with brain engaged and eyes open. Organizations that understand LSS and use it don't try to run 3-minute miles when the world record is 3:40!

Facts Trump Wishes and Opinions

LSS, more than anything else, is about **Managing by Fact**. Constructive solutions always rest on getting, analyzing, and acting on facts – *not* on fault-finding, finger pointing, or exiles to the gulag.

When appropriate values of relevant variables are not known, often the case in the early stages of LSS deployments, it is at least possible to consider a range of worst / best / most likely values and with that information it is possible to model the range and probability of potential outcomes using techniques such as Monte Carlo simulation. The dynamics described above have been extensively documented in the literature for at least 20 years!

We are, after all, always we dealing with uncertainty – every decision in the world we operate in involves risk. Models won't ever give us the absolutely correct answer, but experience shows they get a lot closer than guessing!