## Why Does IT System Development so Often go Awry?
## Trade-offs Among Requirements, Complexity, and Cost

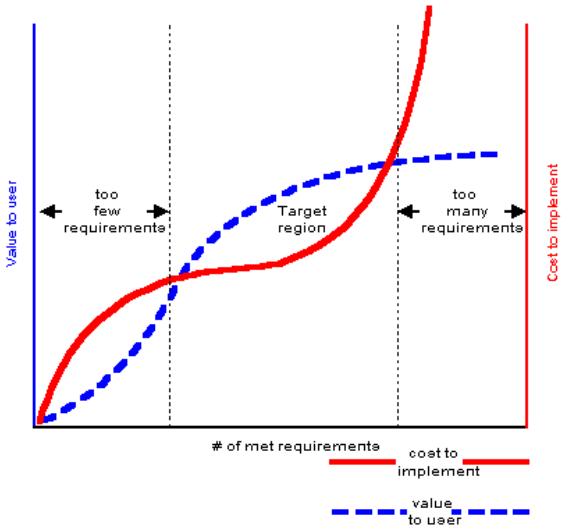
Systems that meet relatively small numbers of requirements will usually give people most of what they need. (If not most of what they want.) But people, many of whom should know better, insist on having it all, and thus doom themselves to building systems that fail. Either the systems are built and do not perform to expectations, or they never get finished. Why is this so, and how can people be brought to appreciate the "requirements trap"? The problem exists because:

The greater the number of requirements, the greater the system's complexity. The greater the complexity, the harder it is to anticipate problems, to test the system, and to fix whatever problems are discovered.

- The greater a system's complexity, the greater the probability that its designers will have to work at or beyond their outer limits of expertise.
- The greater the number of requirements, the lower the likelihood of getting by on commercial off the shelf software (COTS). Once COTS is abandoned, so too are many advantages. 1- System testing and industrial hardening is limited, i.e. there is a decrease in how proven the system is at the time of its deployment. 2- The software is not backed by a company with a financial incentive to maintain and upgrade their product. 3- Costs go up because they are amortized over a fewer number of users, thus putting limiting pressures on testing and documentation. 4- The size and depth of the technical support base decreases.
- Even when COTS is available, the greater the number of requirements, the greater the amount of customization that will be needed.
- As requirements and complexity increase so does the need to coordinate development activities among ever larger numbers of people and groups. The greater the burden of coordination, the greater the likelihood of error.
- The greater the number of requirements the greater will be the system's complexity. As complexity increases, so do the possibilities for non-linear and/or subtle interactions. In essence, system behavior becomes more difficult to predict.
- As requirements increase so does the time it will take to build a system. The greater the development time, the greater the likelihood that new demands will be made on the developers. Thus a vicious cycle is set up making it ever more difficult to complete the project

## The Solution

The essence of the problem is that when specifying a system, people do not appreciate the consequences of what they are asking for. A possible solution is to put them through the exercise depicted in the accompanying figure. In that graph the left "Y" axis plots the value of the system to the user. The right "Y" axis plots the cost of system development. The "X" axis lays out requirements in decreasing order of importance. Ideally real dollars would be attached to the "Y" axes, but the exercise works even if the scale ranges simply from zero to "very high". It works without quantitative data because people do have a general sense that value and costs rise with number of requirements met, and that costs in the "very high" region are worthy of a careful look.




The moral of the tale is told by the plots. When very few requirements are met, a system is going to cost more than it is worth. This is because however it is designed, some foundation costs will be needed, and the ROI is negative when the costs are amortized over only a very few requirements. At some point value begins to exceed costs. This point is about where the dynamics of the 80/20 rule begin to kick in. Past a certain point, though, value increases only minimally with each additional requirement that is met.

Unlike the value curve, the cost curve does not reach an asymptote. Rather, it hits a point where the witches brew of complexity, lack of resources, limits on expertise, time pressure, and coordination begins to bubble. As control of system development is lost, costs escalate exponentially. Its not long before costs exceed value by a very large amount.

Nobody knows the real shape of the curves, but it does not matter. (Of course it would be great to have a chance to do some empirical research on the matter, or at least to build a simulation.) The point is that these shapes make intuitive sense to people who have even a modicum of experience with specifying or building systems. I think they would all agree that something like what is shown in the graph reflects reality. When developing systems, designers should put users through a three step exercise.

- Order requirements as shown on the "X" axis.
- Estimate the inflection points on the value curve.
- Plot the cost curve. (Designers and vendors will probably have to help with this step.) Finally, establish rough estimates for both "Y" axes.

This exercise might prove difficult for people, particularly when it comes to estimating costs. But even with rough estimates, people will begin to realize what risks they are

taking when specifying requirements-rich systems. The goal of the exercise is to get users to specify a system that is somewhere within the "target region" on the graph. It does not matter precisely where, because anywhere in that region represents a reasonable trade-off of cost and met requirements. As for those who still want more, they can rest assured that once the system is built, a continuous improvement process can be used to climb the requirements curve. Also, the experience in building the system in the first place is likely to provide experience that will lower the cost curve, thus making it practical to climb even further up the requirements curve.