My Fair Estimate

The room is full of tension. White boards are plastered with convoluted notes etched in multi-colored dry erase ink. Walls are awash with diagrams (affinity, fishbone, entity relationship, and state), charts (flow, Pareto, PERT, and GANTT), structures (breakdown, data, and control), and lists (personnel, resource, and equipment). Coffee is cold, tempers hot, discussion long, forbearance short, donuts fresh, and ideas stale.

This quotidian scene subsists in software war rooms far and wide. At the advent of a new customer, project, or requirement, managers marshal troops to answer two very elusive questions: how long and how much? These simple questions set in motion conjecture, machination, negotiation, and arm wrestling that would nauseate Johnny Cochran. For all our vaunted powers of ratiocination, software engineers tend to be a fickle lot when it comes to estimation. Why?

Being masters of our domain and desiring to be worthy of the vaunted title of engineer, we ignore the fact that our estimates are inherently subjective. For the past decade, software's leitmotif is that software development is analogous to industrial manufacturing. The analogy hints that software construction can be shaped into a repeatable process where programmers are cogs in a Fredrick Taylor production line. While similarities exist in some areas of software development, estimation is not one. The theory cloaks the software estimation process with a farrago of formal notation and hints at objectivity.

In manufacturing, repeatable and codified processes lead to objective measures and estimates. Software development, on the other hand, is an intrinsically creative activity that differs each time code is manufactured. What you composed on your last project rarely translates objectively to your subsequent project. It resembles Bob Fosse's chorus line more than Fredrick Taylor's production line.

Before the maturity pundits kvetch like contumacious sports stars to impugn my opinion, let me explain. I concur that mature organizations are using repeatable processes, but I contend that the complexity of each project varies. In developing software for the F-16 Head-Up-Display, I used the same process and techniques to construct the "Altitude Low Warning" module and the "Enhanced Envelop Gun Site" module. Yet the complexities involved in constructing those two components were about as close as Bill Gates and Larry Ellison. Bollinger elaborates this point in his *IEEE Computer* article "The Interplay of Art and Science."[1]

Variation in complexity, which is difficult to objectively measure, dominates a software project estimate. Consequently, estimating software is unavoidably subjective. Therefore, as we estimate our projects, instead of emerging as the professor of estimation we end up more like Gilligan.

Second, we prefer precision to accuracy. Software engineers favor specific single-value estimates, which are certain to be wrong, over a range of values that have a high probability of enclosing the correct estimate. This concept should not be foreign; we use it all the time. When a spouse asks what time we will be home, we always give a range because we know that if we answer 4:12 p.m. and waltz in at 4:15 p.m., we are sleeping on the couch.

Then there is the Pygmalion effect? From Greek mythology, Pygmalion was a king of Cyprus who carved and then fell in love with a statue of a woman. Psychologists Robert Rosenthal and Leonore Jacobson attached Pygmalion's name to the observation that when evaluating something, the evaluator is hardly neutral, and the evaluator's expectations influence the evaluation.

This was personified eloquently in Bernard Shaw's play "Pygmalion" in which phonetics professor Henry Higgins tutors a Cockney flower girl, Eliza Doolittle, in the refinement of speech and manners. For those who avoid the theatre you may have caught the story in the musical "My Fair Lady?" If you are still not with me, join the theatrically impaired and visualize "Pretty Woman" with interesting dialogue and wit.

In this yarn, the project at hand is the transformation of Eliza into a lady. Participants in this transformation are: Professor Higgins who, despite his love of Eliza, can never truly commit himself fully; Freddy Hill who is naively infatuated with Eliza; and Colonel Pickering who seems aloof of the antics but always seems to be there at the right time with the right words.

How does this apply to estimation? Stakeholders are about as focused on estimation accuracy as my son is on picking up after himself. They provide specious estimates to impress clients and, like my son, are improvident to the mess they leave behind. Stakeholders are more callow than a freshman engineer at a fraternity party. They, like young Freddy Hill, are in love with a project's prospects with little concern for the consequence of their credulous desires.

Software engineers, like Professor Higgins, are more than willing to demonstrate their knowledge, wisdom, and prowess but are short of committing to the minutiae of the project's long haul. We are prone to embellish the estimate to assure that our reputation, health, and marriage remain in tact.

Estimations involving human intervention are prone to the Pygmalion effect, and software estimation is no exception. Exuberant stakeholder expectations counter engineers who, if the truth were known, fervently wish they could get home from the office earlier and come in less on weekends. The fact is no one is a good (impartial) judge of one's capability because our perception of the problem causes bias.

That's where Colonel Pickering comes in – a prudent counselor who mixes analysis with common sense. A sage that applies experience, intuition, and judgment to obviate subjectivity, employ flexibility, and temper bias. A team needs a Pickering to mediate between stakeholders and engineers and swing back the estimation pendulum from fallacious exactitude to viable accuracy.

Higgins forewarns, "… you can come back or go to the devil: which you please."

— Gary Petersen,
Shim Enterprise, Inc.

1. T. Bollinger. "The Interplay of Art and Science in Software ." *IEEE Computer* Oct. 1997.