EMBEDDED IN THIN SLICES

Estimating Your Embedded Systems Project (Part 3)

Four Heuristics for Embedded Software Development

Bob concludes his series on estimating the costs for designing and developing your embedded systems project. He looks at four heuristics and how by knowing them you can get better at this task.

By Bob Japenga (US)

"People who spend their time, and earn their living, studying a particular topic produce poorer predictions than dart-throwing monkeys." This quote from a Nobel prize winner who knows his stuff is a jarring introduction to our final installment in our article series dealing with estimating the cost and schedule of our embedded software systems. Is this whole process of software estimation no better than what dart-throwing monkeys could come up with? In the opening article, I said that accurate estimating is extremely difficult. But I also said that there is some hope. This month I would like to provide some thin slices of help for anyone who is asked to estimate how many man hours it will take to create an embedded system or even a part of an embedded system. And the help will come from the author of that quote.

Daniel Kahneman is a psychologist who won the Nobel Prize in Economics in 2002. The above quote is taken from his 2012 book entitled *Thinking, Fast and Slow*, which summarizes his decades of research on the psychology of judgment, decision-making, and behavioral economics. Kahneman has provided some key insights into how we approach the impossible task of estimating costs of developing embedded software systems.

This article will not delve into function points, use-case points, software metrics, COCOMO models, SEER-SEM, or any of the other methods for estimating software. These are good, useful, and well documented in the literature. We use function points, use-case points and software metrics in our company. I would heartily recommend that you understand function points, use-case points, and develop software metrics. In particular, develop software metrics that relate to your experience of function points or use-case points. In other words, count the function points or use-case points during the estimation phase of three to four projects and see what you learn about yourself, your estimating process, and your projects. Go back to completed projects and determine the number of function points or use-case points and factor that into a metric.

What I want to do this month is to look at estimating from 5,000′. I want to see how

Kahneman's research can help us become better at estimating software. The stated purpose of his book was, in his words, to "learn to recognize situations in which mistakes are likely and try harder to avoid significant mistakes when stakes are high." If we can glean that from his book, we will become better at estimating the costs of embedded software systems.

Kahneman proposes almost 50 heuristics in his book. A heuristic is a method or process that enables us to learn something on our own. Only a few of them are applicable to us in estimating software. But if we can master them, they will enable us to become better at estimating embedded software.

PRIMING

Sometimes one of our customers will tell us that a project needs to be completed in three months. Or that it needs to be completed for under $15,000. These numbers can have a very bad effect on our ability to accurately estimate a software project. I am amazed how often my estimate closely parallels the customer's estimate. Can it be that the customer really knows how long it is going to take or how much it is going to cost? Or is something else going on?

A heuristic discussed by Kahneman in his book is called priming. He and other researchers have demonstrated that our behavior can be primed by what goes immediately before us. For example, he sites one study, where two groups of young people (aged 18–22) were asked to form some sentences from a set of five words. One group had a set of five words associated with the elderly. After the exercise, the young people were asked to walk through a corridor. Those who worked with words about the elderly walked slower than the other group! Experiments like this have been repeated many times with a wide variety of different priming mechanisms. The evidence seems to indicate that we are deeply influenced by priming.

How are we to use this knowledge about ourselves to become better at estimating? First, as much as possible, we need to avoid obtaining from our customer or bosses expected costs and schedule before we make our estimates. Estimating is difficult and I really want to know what the customer or my boss expects me to estimate. But resist the urge. Priming is a powerful and proven effect and we must avoid it as much as possible. Watch out when your boss tells you that he needs this in two weeks and then asks you to estimate it.

If however, the cat is out of the bag, we need to make an extra effort to not let that number influence us. This is by far the harder of the two options. Once primed, even when I take herculean strides to not be influenced, the priming has its effect. But at least I am aware of the effect. Develop your estimate with your usual method of function points or use-case points or whatever, and if it comes in the same ball park as the "primed" number, be wary of your numbers and extra-vigilant—run your numbers again.

ANCHORING EFFECT

I have noticed that a lot of my estimates seem to be remarkably similar to previous estimates. Could it be that my projects are so similar that it always takes 40 hours to write the software specification for all projects? Or that user interface designs always take 160 man hours. Or is something else at work?

One of the heuristics Kahneman has identified is what he calls the anchoring effect. The anchoring effect "occurs when people consider a particular value for an unknown quantity before estimating that quantity." With serious academic rigor, Kahneman demonstrates how we are influenced by previous numbers. For example, he tells us that if we were asked if we thought that Gandhi was 114 years old when he died, we would immediately say "No." If we were then asked how old we thought he was when he died, our number would be higher than if we were first asked if we thought Gandhi was 35 years old when he died. That first number acts as an anchor to pull our estimate in its direction. Kahneman's claims that this phenomenon is "one of the most reliable and robust results of experimental psychology: the estimates stay close to the number that people considered [previous]—hence the image of an anchor." This means that we will be affected by it when we do our estimating.

Anchoring is closely related to priming. I would make the distinction that priming involves numbers related to the estimate (the customer's estimate for the same project). Anchoring happens when I take numbers from an unrelated project into account before I make my estimate.

How can we take this knowledge and become better at estimating? Here is where software metrics come into play. Look back over the last several estimates of unrelated projects. Were the estimates similar to each other? How did the actuals compare to the estimates? If you see a correlation with the estimates but not in the actuals, anchoring is a possible cause. Develop a range of estimates based on actuals and use these when making a new estimate. For example, imagine that the user interface took 120 hours on project A, 130 hours on project B, and 250 hours on

ABOUT THE AUTHOR

Bob Japenga has been designing embedded systems since 1973. In 1988, along with his best friend, he started Micro-Tools, which specializes in creating a variety of real-time embedded systems. With a combined embedded systems experience base of more than 200 years, they love to tackle impossible problems together. Bob has been awarded 11 patents in many areas of embedded systems and motion control. You can reach him at rjapenga@microtoolsinc.com.

project C. Attempt to identify the similarities and the differences and place a weighted value to each. How many menu items or screens were involved? Was it a graphical interface? Was a working driver provided? Was it a touch screen or keypad or both?

To avoid the anchoring effect on new estimates, we need to ruthlessly dissect previous projects into their subcomponents using software metrics. In other words, keep track of how long it took to write the specification, design the user interface, design the manufacturing test fixture, etc. Then when we approach a new project we need to make a quantitative comparison between the smaller elements. For example, the user interface is about twice as complex as project B and half as complex as project C. This quantitative approach can help us avoid the anchoring effect.

OPTIMISTIC BIAS

Kahneman describes many biases that affect our ability to estimate. He posits that the optimistic bias may be the most significant. In my earlier articles in this series, I discussed optimism as it relates to estimating but it bears repeating. I would recommend reading chapters 23 and 24 of Kahneman's book in an attempt to hammer home how pervasive this heuristic is and learn to make adjustments.

How do we counter this optimistic bias? I would say that a thorough knowledge of our optimistic bias is a good start. Software metrics can help if you develop actual numbers and then compare them to your estimates during a post-mortem. But human nature such as it is, unless we learn to develop a sort of "humility before the data" we can ignore the stubborn facts the metrics show us. A simple mantra that could be said after you have completed your estimate and before you submit it, is to repeat these words: *All evidence shows that I am repeatedly over optimistic in what I think I can do. How should this estimate change based on that?*

SMALL SAMPLE SIZE

We all know that using a small sample of data sets us up for errors in estimating or drawing conclusions. But Kahneman takes us to a new level of awareness of the danger of using small samples. For example he cites a study of the incidence of kidney cancer in 3,141 counties in the United States. The counties with the lowest incidence per capita are "mostly rural, sparsely populated, and located … in the Midwest, the South and the West." Upon hearing this, most of us immediately start jumping to conclusions. But he goes on to also state that counties with the highest incidence per capita are also mostly rural, sparsely populated, and located in the Midwest, the South and the West. I leave it to you to figure out why sample size is the reason for this apparent contradiction. Email me if you want some help.

In using our software metrics to estimate embedded software systems, we have to recognize that we have an extremely small sample size that we are drawing upon. I have been in this business since 1973. I have estimated almost a 1,000 such projects. Yet even with all that experience, that is an extremely small sample to accurately predict how the next project is going to go.

So what can be done in light of this last heuristic? I recommend that you doggedly pursue from other companies the results of actual projects. Most companies are not willing to part with this information. But I have found that it doesn't hurt to ask. In non-competing situations, we can learn a lot as we expand our sample. There are a lot of numbers floating around the web. Take the time to create your own data base of "the other guy's" actual development time.

On one project, we had expended an immense amount over our original estimates. After the project we found that another company designed a very similar product and took 2× to 3× as much calendar time and cost. Had we had that number in the beginning, we may have been more accurate in our original estimates.

Although your environment is unique to you and your company, industry standard metrics like hours/line of code and hours/function point or hours/use-case point can help expand your sample. These metrics are prone to many errors and are widely variable. Nonetheless they are another data point for your estimate. They help us limited engineers do the impossible: expand our sample without actually doing the work.

HOW ACCURATE?

Accurately estimating embedded software systems is impossible. Don't let anyone tell you otherwise. Hopefully, with some input from this series, you will become a little better at it than you were before. And that is no small accomplishment. It is with some reluctance that I close this article series since

circuitcellar.com/ccmaterials

RESOURCES

D. Kahneman, *Thinking, Fast and Slow*, Farrar, Straus, and Giroux, New York, NY, 2012.

I know that I have taken a very thin slice of a very big topic. Email me if you would like me to elaborate more fully on any of these topics in the coming months.