Truth and Confidence: Some of the Realities of Software Project Estimation

Phillip G. Armour
Senior Consultant Corvus International, Inc.

Software project estimation is not what we think it is because, to some extent, software is not what we think it is. This article explores an alternative view of both software and project estimation and concludes that the process of estimation could be much more valuable than we usually make it.

Predicting the future can be a rewarding occupation, but it can also be a dangerous one. Historically, oracles were often praised and lauded. But they were also stoned to death if they were wrong – and sometimes if they were right [1].

Software project estimation is a difficult task for a simple reason: Software is not really a product, it is a packaging of knowledge and we cannot measure knowledge. Software is best thought of as a knowledge storage medium rather than a manufactured product [2]. It is one of the five places we can put knowledge once we have obtained it, the other four being (in historical order): DNA (Deoxyribonucleic Acid), brains, hardware, and books. However, knowledge in software has different characteristics than knowledge stored in the other media [3].

Shooting Tanks

During World War II, the knowledge of how to hit a tank with a bazooka was stored in several places: It was stored in military manuals (the book form) and in the ranging and sighting device (the hardware form), but it was stored mostly in the operator's head (the brain form) (see Figure 1). There are some drawbacks with these media: The manual only *describes* the knowledge – it does not actually do anything; the sighting mechanism allows for storage and use of only a few of the variables – mostly the distance-to-target versus elevation relationship; and the brain-resident knowledge has the distinct disadvantage that the soldier could be shot at while attempting to hit the target. In modern weapons systems, we have moved almost all of this knowledge into the missile in an active software-resident form.

Not Product Producing

If software is not a product, but is a medium, then software development is not a product-producing activity. In fact, it is best thought of as a *knowledge acquisition* activity. Most of the effort on a software project is related to acquiring and validating knowledge rather than creating a product. We know what we are doing.

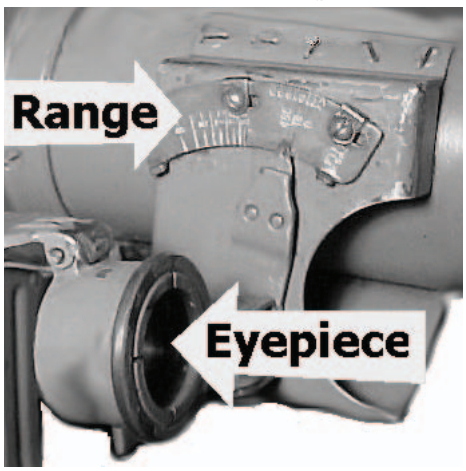
In an attempt to estimate projects, we are trying to figure out how much knowledge we do not have and how much time and effort it will take to get it, plus a small amount of time and effort to translate it into the executable form once we have obtained it. There are two challenges to this: First, we are trying to measure something we *do not* have which is always hard to do, and second – and very importantly – we are trying to measure knowledge, and knowledge is simply not a measurable thing.

This leads us to some observations about the essential nature of project estimation:

- **We cannot have an accurate estimate.** Apart from it being an oxymoron, there is a simple reason why estimates cannot be accurate – we simply do not have the data or knowledge we need to be accurate. The primary activity of a software project is to get this knowledge. The only point in time where we can reasonably assert we are accurate is at the end of the project when we have acquired all the knowledge and resolved all the uncertainty.

  It is possible to have a *lucky* estimate. This happens when all the things we did not or could not think of that slowed the project down and all the other things we did not think of that speeded the project up happen to be equal. Since there are more things that will slow a project down than speed it up – an application of the 2nd Law of Thermodynamics to projects – we usually underestimate.
- **The purpose of estimating is not to come up with an end-date.** This is usually what we are asked for when someone wants an estimate, but it is not a well-formed request. For most projects there is a wide range of possible dates when the project might finish (see Figure 2, page 28). At the point in time when we produce the estimate, we can posit a trade-off of *probability of success* for schedule and other resources. It is easy to be 100 percent successful in projects – simply take a very long time and use a very large number of very good resources. In reality, the purpose of estimation is not to deduce an end-date, it is to derive the *probability function* that describes the range of viable end-dates. The project completion date and schedule is not determined by estimation but by the commitment process.
- **Estimation is not commitment.** Making an estimate is not the same thing as making a commitment. The job of estimation is to identify the project's probability function. The job of the commitment process is to select the point along the probability function that best manages the risk/return ratio. Estimation is a technical activity; commitment is a *business* activity, and they operate on quite different data.
- **The project estimate may not be dependent on the delivered system size.** Despite the fact the every estimation process used in software development operates on the expected delivered system size, the relationship can be quite tenuous. The final system size may be an *indicator* of the effort necessary to develop the system. All else being equal, if one system expects to have twice as many (say) lines of code as another, that

Figure 1: *Bazooka Sighting Mechanism. Photo property of <www.antiquefirearm.com> and <www.andrax.com>. Used with permission*



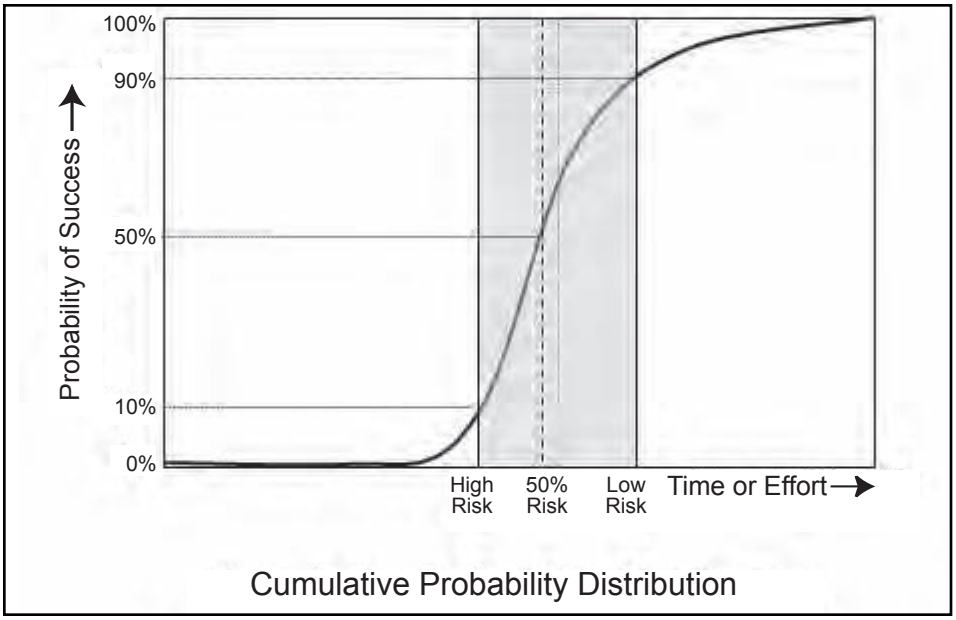


Figure 2: *Probability of Completion*

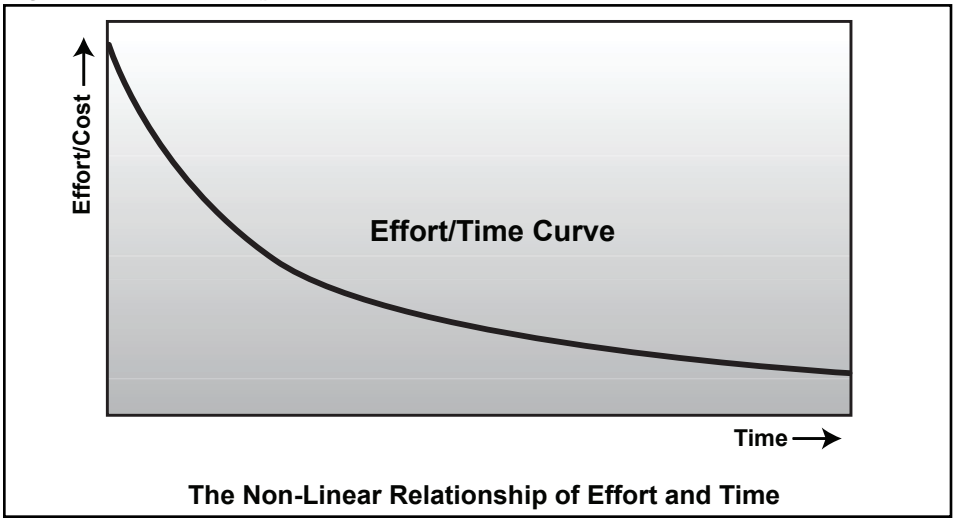
system will require proportionally more time and effort. The trouble is, all else is rarely equal. Viewed as a knowledge acquisition activity, it is clear why project effort may not have much to do with the final size. If we use experienced developers, they do not have to acquire as much knowledge to produce a system of a given size as less experienced developers, but the system size does not change. If we can reuse code, either from a library, or embedded within a language, the effort is less, since some of the knowledge is already stored in an accessible software medium.

Also, a system may be small in terms of its line-of-code form, but it may have very high *knowledge density* as is true of real-time embedded systems – the amount of knowledge needed to make them work divided by the final executable size is much higher than for typical business systems. However, while the knowledge density of an information technology system might be light, it may constantly change as the market changes, meaning the knowledge must be reacquired. So, the effort, schedule, staff and cost may increase or decrease without respect to system size at all.

Almost all estimation processes and tools provide a way of *tuning* the final-size-driven estimate by adjusting parameters which represent the systems' attributes. We also trust these adjustments will track to the effort necessary to acquire the knowledge. The net result of this tuning may entirely submerge the effect of the system size.

- **The effort-time relationship is not linear.** In fact, it is a high order reciprocal exponent [4]. It is common for organizations to believe that the process of building software is, well, a building process rather than a knowledge acquiring process. Accordingly, they operate on a set of assumptions based upon manufacturing. This includes the relatively linear relationship of effort (people, machines) to time to deliver. In a factory, if we double the number of machines or run the machines twice as long or twice as fast, we will approximately double the output. This math simply does not apply to software because it is not a manufacturing process. It also partly explains why adding people to a project – particularly when it is already running late and the schedule is already compressed – is not an effective tactic (see Figure 3).

Figure 3: *High Order Reciprocal Power*



The Job of Estimation

The real job of estimation is not what it seems. True, it does have a very important role in determining the basic planning parameters of staff size, project duration, effort and cost (and for some estimation models, quality and defects). This is the classical *tell me when the project will be done* role of estimation.

The calibrations necessary to achieve a useful, as opposed to accurate, answer from an estimate are complex. They *characterize* things: the system being built, the environment and people working on the project, the management of that environment, the tools and their effectiveness, the level of documentation, and many other factors. This is clearly seen in the operation of many parametric estimation tools and processes. For instance, the COCOMO II model uses Scale Factors which determine the size exponent. These include the following:

- System/project attributes: How new the project is.
- Project/process attributes: Development process flexibility and architecture risk resolution.
- Team attributes: Team cohesion.
- Organizational attributes: Process maturity [5].

COCOMO II further expands on its characterization with a set of *cost driver* factors which reflect everything from the documentation level to the *complexity* of the system being built.

Given a reasonably sound characterization of an environment and system using these types of factors – a very significant task in itself – an estimation process becomes an analogue of the projects being run in that environment.

A Certain Uncertainty

Even if we think we are able to *accurately* size a projected system, and we have good data which we think characterizes the team, sys-

tem, and environment, we invariably find that these are not certain. With each factor, there comes some degree of variability: We may have been *this* productive in the past, but our productivity may be different now. The project might be *this* big, but it could be that big. It could be on the simple side of complex, or the complex side of simple. The new tools or language we are planning to use might help a lot, a little, or they might require more effort than they save. The only way to be truly certain is to try it.

For each of the factors, we can postulate that they will likely operate over a range of values. The product of all these variances determines the aggregate uncertainty of the project and defines the slope of the cumulative probability S-curve in Figure 2. There are many challenges to calculating these ranges, not the least being that the factors are not independent. Processing the individual variances in a statistically legitimate way allows us to calculate the *total uncertainty* in the final project solution(s).

Estimation as Simulation

Here we get to the real purpose of estimation. If we have reasonably characterized the environment, if we have established some operational variance for the size of the system and its complexity, if we have some idea of the ranges of difficulty in obtaining the knowledge for the system, and if we have some calibrated way of processing this information, we could *simulate* what might happen when we run the project.

The concept of a statistical approach to the management of software (especially under the umbrella of Six Sigma) has its detractors and they have some very good points [6]. Developing software is not the repetitive cranking out of identical units. Indeed, doing something differently from the last time is a bad thing from a manufacturing perspective, and most of the effort in statistical process control is dedicated to identifying, analyzing, and removing variance. But in software, variance is the reason why we have a project at all – if we wanted to do it just like the last time, we would simply use whatever we produced last time. Again, software is not a product at all, it is a medium in which we express, store, and make active the knowledge that we gain when we run a project. It is the knowledge that is the thing, the software is simply where we put it.

We do not have the resources to run the same project multiple times to see how to run it best. We only get to run a project once. However, we could set up an estimation system, with certified and controlled inputs, that reasonably collects the likely variances in the key characterizing factors of product, personnel, technology, and environment, and we could model the interaction of these variables reasonably well. Doing this, we can simulate the behavior of our organization when it runs the project under a given set of conditions. These results can only be expressed in probabilistic terms – since we have uncertain inputs, we must have uncertain outputs. These outputs look a lot like statistical variance analyses, even though we only have one project. Many companies have and use various estimation processes and tools, but few have established estimation *systems* that control, audit, and report project estimation and risk data in the same way we control, audit, and report on accounting data.

The financial management sections in most companies have financial models that they create, manage, and use. These models incorporate key factors in the financial markets: inflation, growth in Gross National Product, cost of capital, market sensitivity, etc. Companies find these tools very valuable in helping to understand what kinds of decisions might be more optimal than others. Do these tools predict the future? No. They cannot do that. But they can and do help in the financial management of companies; they are very valuable tools and systems. We could do the same thing for software projects and estimation.

Comedian Woody Allen once remarked "the only thing I cannot accurately predict is the future…". What an estimation simulator could do is help identify more (or less) optimal decisions about how we run our projects, before we actually run them. We often teach pilots on simulators. They do not replace learning on the real thing, but you can try things and test out behaviors on a simulator that you would not want to try on the real thing.

Fifteen Too Many, One Too Few

Several months ago, a client of mine was considering implementing a large project in 15 equal increments spread over three years. Using estimation tools to model the whole system in a one-release, three-year delivery, *big bang* approach we showed that this was not a highly constrained system. However, modeling the 15 increments in our estimation system showed that the sum of the parts was a lot bigger than the whole. We could demonstrate that the overlapping increments inserted a very high degree of risk into the project that would only become evident some way down the line. This project would look pretty good for about two years and then the wheel would fall off. The big bang approach was much less risky, but the customer would see no value for three years. We modeled many possible solutions including a four, unequal-increment solution that we were able to demonstrate would deliver the most functionality to the customer at the earliest date, with the lowest risk.

We could have learned that the 15 increment solution was a bad idea by trying it, thereby costing the company several million dollars. Or, we could learn the same lesson by simulating what would happen and preemptively picking a more reasonable course.

There is little doubt we need to improve our performance in software project estimation. The stakes are very high, but if we align our expectations of estimation in accordance with the reality of software development and set up our organizations to feed a software development business simulation system, the rewards will be even higher. We can do that.◆

References

1. Wood, Michael. The Road to Delphi: Scenes From the History of Oracles Farrar. New York: Straus and Giroux, 2003.
2. Armour, P.G. "The Case for a New Business Model." Communications of the ACM 43.8 (2000): 19-22.
3. Armour, P.G. "The Laws of Software Process." Boca Raton, FL: Auerbach Publishers, 2003.
4. Putnam, Lawrence H., and Ware Myers. Measures for Excellence. Englewood Cliffs, NJ: Yourdon Press/Prentice Hall, 1992.
5. Boehm, Barry, W, et al. "Software Cost Estimation with COCOMO II." Upper Saddle River, NJ: Prentice Hall, 2000.
6. Binder, Robert V. "Can a Manufacturing Quality Model Work for Software?" IEEE Software 14.5 (1997): 101-105.

About the Author

Phillip G. Armour is a senior consultant for Corvus International, Inc. He is a contributing editor at *Communications of the ACM* and authored the book "The Laws of Software Process."

Corvus International, Inc.
205 Briargate LN
Deer Park, IL 60010
Phone: (847) 438-1609
E-mail: armour@corvusintl.com