

## Real Work, Necessary Friction, Optional Chaos

Challenges in estimating software scope by effort.

**W**hen I first came to the U.S. and relocated to the Chicago area, I asked someone how far away was Madison, Wisconsin. “About two-and-a-half hours” was the reply. This puzzled me, since the unit of “how far away...?” is miles; hours is the unit of “how long will it take me to get to...?” We routinely convert miles into hours for travel, and sometimes we perform a similar conversion when we produce estimates for software development.

I recently encountered this when estimating a project. The question “how big is this system you propose to build?” elicited the response “...about 10,000 staff hours.”

Measuring the “size” of software is very difficult. The core problem is that we are trying to measure knowledge, and it is simply not empirically measurable. This is compounded by the fact that the effort and schedule of a project are driven, not by what we know (Zeroth Order Ignorance—0OI) or what we know we don’t know (First Order Ignorance—1OI) which we might have some hope of measuring, but by what

we don’t know we don’t know (Second Order Ignorance—2OI) [1]. Since 2OI tends to be the most effort-intensive component of a project it is the most important factor in estimating and anyway we don’t even have a



definitive unit in which to express a system size [2].

One could argue that sizing a system by effort might actually make more sense than sizing it by some size metric like “Lines of Code” (LOC)—after all, it is the effort that largely defines the cost. The effort divided by the resources available usually deter-

mines schedule, and few customers know or care how many LOC are in the system. Additionally, most size-based metrics such as LOC don’t explicitly address important development attributes such as how difficult it is to acquire the knowledge necessary to create the LOC, which an effort metric clearly does. Given that we really can’t measure the “size” of a system directly (though it can be reasonably inferred in a number of ways), what is wrong with expressing system magnitude in staff hours or person-months?

### The Magical Disappearing Effort

There are two issues with doing this, both of which surfaced during this project. The first is that available software estimation tools do not directly process an input of effort—effort is a primary *output* of such tools. The tools generally require the expected system scope to be expressed in a size metric such as LOC. The second, more subtle, issue is that effort is not invariant (neither is LOC, but that is an issue for another day).

Let me explain.

In defining the “size” of the system in question, several experi-

# The Business of Software

enced project managers determined that it should be around 10,000 staff hours. However, in processing the data available to produce an estimate for the actual effort, schedule, and staffing levels, we were presented with what appeared to be a paradox. If we decided to accelerate the schedule for this system, it was fairly obvious and quite acceptable that it would, in fact, require a lot more than 10,000 hours of effort. We all understand it is not possible to deliver such a system in one hour using 10,000 people. Accelerating the schedule to one-half of the “nominal” time does not just double the number of people. The relationship between effort, time, and people is not linear. In fact, one of the most popular and effective tools on the market models an effort relationship of  $\text{Time}^{-4}$  [3] (see Figure 1; the SLIM-Estimate<sup>®</sup> tool created and marketed by QSM Inc, McLean, VA, expresses a  $\text{Time}^{-4}$  relationship of effort). This reciprocal fourth-power produces some jaw-dropping results when a schedule is compressed, and it shows just how expensive it really is to deliver a system before its time. But there are two sides to the coin. If we have the business degrees of freedom to intentionally postpone the delivery of the system, we can save substantially in cost and resources. In fact, deliver-

ing the same 10,000 hours worth of system somewhat later than the nominal time might cost only 6,000 hours of effort—how can this be?

How is it possible to deliver the same functionality for 60% of the initially expected cost simply by taking a bit longer? The answer

lies not in the functionality, but in the friction and the chaos on the project. For those components or activities on a project where we do know what “right” is, there is a certain amount of effort necessary to factor our knowledge into an executable product. This work obviously goes directly into adding knowledge to the product or, more correctly, translating the knowledge we already have into an executable form. This is what I call “Real Work.” Real work is a product of the size, complexity, and functionality in the final system. It is reasonably well indicated by size-based metrics such as LOC and Function Points, and is generally a fixed value that does not vary with the time taken to develop the system.

## Necessary Friction

But what if we don’t know what is “right?” Usually, when we begin a project, we do not

know everything we need to know to create a functioning system. One of the tasks of the project is to discover this knowledge and then to factor it into an executable form. As any experienced developer will acknowledge, it is the finding of the knowledge that is the difficult part. The mechanism for doing this is the project’s development methodology and process. The extent and difficulty of finding this knowledge also guides the selection of the life cycle and approach to its discovery.

For systems with high levels of 2OI (new customers, new functions, new technology, and so forth) we would typically adopt an

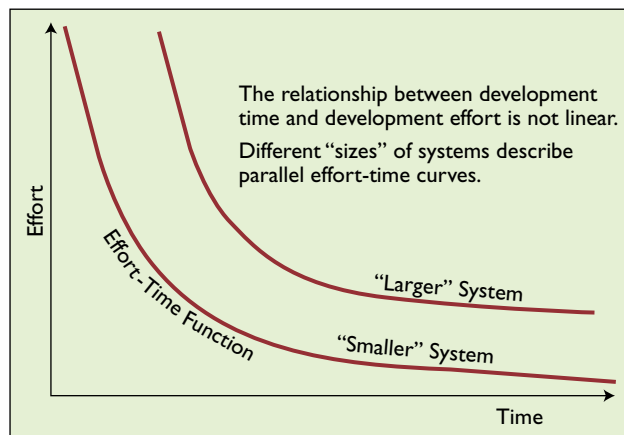


Figure 1. Effort-time function.

lies not in the functionality, but in the friction and the chaos on the project.

## Real Work

Only some of the effort on a project actually goes into the final product in any real sense. As I have asserted in previous columns, it is almost never possible to “do it right first time,” since if it really is the first time, we know neither what “right” is nor how to do it. For those components or activities on a project where we do know what “right” is, there is a certain amount of effort necessary to fac-

“exploratory” approach such as prototyping or a spiral approach [4]. When we do this, we understand that a significant portion of the work will not go directly into the product we ship to the customer—the prototype effort is meant to uncover the knowledge necessary to build the functioning system, not to actually build it.

Even in projects that do not require explicit prototypes, some effort is always expended discovering things that are unknown at the start of the project.

Except in very rare situations (where the project consists entirely of 0OI and 1OI) it is not possible to transcribe the knowledge flawlessly into the functioning system during the first attempt. The effort required to experiment, explore, and authenticate the necessary knowledge is not a one-pass action. I call this work “Necessary Friction.” It is the work required to discover the unknown knowledge for the “Real Work.” Necessary friction is a function of the degree of 2OI, the difficulty of obtaining the knowledge, and its complexity. It is only partly related to the scope of the system. The necessary friction component of work is also somewhat sensitive to the time taken to develop the system. It is increased by attempting to complete the

work faster and decreased by taking a longer time. But it cannot be eliminated.

### Optional Chaos

When gathered in numbers and pressed for time, people exhibit a kind of Brownian motion, as anyone who has ever tried to steer a large group of tourists or small

and the distribution of the system’s knowledge across a larger number of people adds many more opportunities for misunderstanding and its associated invalid effort and rework. I call this type of work “Optional Chaos.” It is not “necessary” to the production of the system the way that the real work and the necessary friction are.

Sometimes developer A makes a change that must be communicated to developer B. Developer B responds to the change with additional changes, which must be communicated back to developer A. This causes developer A to make changes that must be communicated back to B, and so forth.

I have seen projects which, with more than enough people and less than enough time, work very hard but actually

achieve very little per capita, with everyone simply reacting frantically to each others’ changes. This effort does have some relationship to the degree of 2OI and 3OI, but it is mostly dependent on the number of people and the lack of time. It is this factor, more than anything else, which drives up the enormous increase in effort when schedules are compressed. Figure 3 shows how a system sized by effort can become “smaller” if we have the liberty to develop it over a longer timeframe.

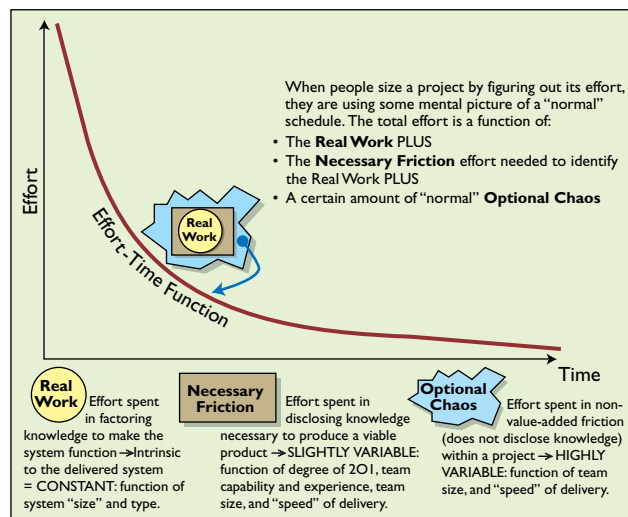


Figure 2. Normal schedule compression.

children will attest. When we attempt to accelerate projects, we introduce a high level of this kind of activity. High-stress projects put pressure on people to make quick, sometimes unvalidated, decisions. The need to increase production to create a system rapidly requires a large number of people. This increases the levels of both communication and miscommunication.

The degree of knowledge across the project is often not consistent,

# The Business of Software

## “Normal” Optional Chaos

When we size a system by the effort we expect to spend in creating it, we must be visualizing some “normal” scenario (see Figure 2). Since effort is never independent of the environment in which the effort is expended we must be thinking, consciously or not, of some point along the effort-time curve where we make our effort assessment. My experience has been that the point location is related to the way an organization usually runs its projects. Some companies are highly market-driven and do whatever they can to minimize the schedule, often at significant cost. Management asking for the “best” (meaning the quickest) estimate is a common symptom of this. Others have ceilings of cost or headcount that bound the problem and provide a minimum time in which the project can realistically be completed. With every project we commit to we are, knowingly or not, signing on to a certain amount of “normal optional chaos.” When people assign a nominal effort for a project in this environment, they must be factoring in all three kinds of work.

## Useful and Useless Work

When asked which of the three kinds of work is most useful, most

people would probably choose “real work.” It is, after all, the real work and determines what the finished system will do. However, we could make a case that, for the developing organization, the “necessary friction” is most valuable

projects sometimes sacrifice productivity on the altar of production, and generate an awful lot of work that benefits no one.

## But How Far Is Madison?

Madison, Wisconsin is not always 2.5 hours from Chicago, because it depends on how fast you drive. But the time of travel is only one factor. Gas mileage is another, and so is the condition of your vehicle. We know and accept that if you drive extremely fast you will use up considerably more fuel in the same distance. You also might just blow up the engine or skid off the road and crash in your haste to make it there just a little faster.

Besides, as anyone from this part of the Midwest will attest, you must also allow for the inevitable road construction along Interstate 90. ■

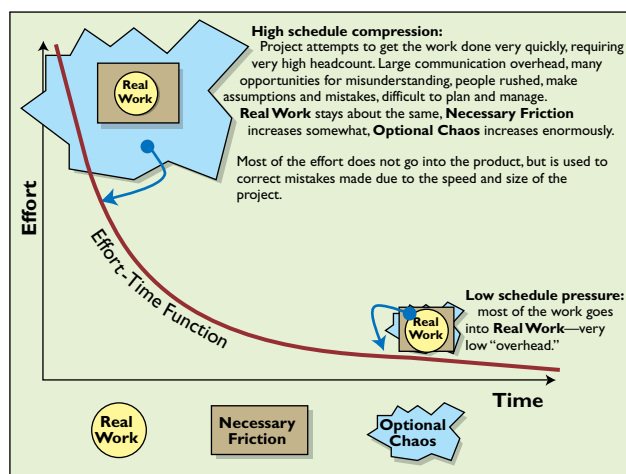


Figure 3. High and low schedule compression.

because this is the new-knowledge discovery work. As in science, the most valuable thing we can do is not catalog what we already know, but discover what we don’t know. Companies that do this are the ones that break open new markets and create the truly killer applications.

Optional chaos is mostly noise, but sometimes it is the greatest amount of work we do on a project. There is an implicit, but erroneous, assumption made by organizations that somehow *all* the work done on their projects is useful and necessary, but this is not the case. Organizations and

## REFERENCES

1. Armour, P.G. *The Laws of Software Process*. Auerbach, 2003.
2. Armour, P.G. Beware of counting LOC. *Commun. ACM* 47, 3 (Mar. 2004), 21–24.
3. Putnam, L.H. and Myers, W. *Measures For Excellence*. Pearson Education, 1992.
4. Boehm, B. A spiral model of software development and enhancement. *IEEE Computer* (May 1988), 61–72.

**PHILLIP G. ARMOUR** (armour@corvusintl.com) is a vice president and senior consultant at Corvus International Inc., Deer Park, IL.