Familiar Metric Management: Small is Beautiful Once Again

Selecting the right team size—*small*—is the key to a successful project. By successful we mean one that comes in on time, on budget, with good quality. Further, we mean one that actually completes faster with less effort than the same project attacked with a large team.

In our column "The Effort-Time Tradeoff—It's in the Data" (March 1997), we showed (in Figure 25) that projects with 20 or more people used a lot more effort (person-months) than projects using five or fewer people. That comparison was for projects of the same size in source lines of code, presumably requiring about the same amount of work. For example, at a size of 50,000 SLOC, the large teams expended 99 person-months, the small teams, 19 person-months. The numbers seem hardly believable, and perhaps some of you raised your eyebrows slightly.

In that early column we were putting the emphasis on the effort-time tradeoff that the software equation implies:

Functionality (size) = (Process Productivity) (Effort/B)$^{1/3}$ (Development Time)$^{4/3}$

In this column, aided by new data on almost 500 projects, from a large corporation, we are putting the emphasis on the great value of small teams. We remembered—somewhat vaguely perhaps— that this subject came up when we were in graduate school. So we hunted around for the most recent textbook on management we had handy. It turned out to be one left by one of our sons when he was in college. After a 13-page chapter, citing 58 references (including the Bible), the authors of this ancient tome concluded:

*". . . where the work is not routine and requires members of the work group to have frequent contact, exceeding the traditional limit of six should be done with caution and serious consideration of the undesirable consequences which may follow." [fn]*Alan C. Filley and Robert J. House, *Managerial Process and Organizational Behavior,* Scott, Foresman and Company, Glenview, IL, 1969, 502 pp.[/fn]

That was written when the software age was still young (1969). The authors' references go back as far as 1927. The idea of small work groups is hallowed by time. If ever there was work that is not *routine* and requires *frequent contact,* it is software development. What we now bring to the table is *metrics.* Those metrics establish that the concept of using small work groups in software development is a *really beautiful* idea. Let us show you the figures.

Small groups complete projects in less time

The data that we obtained came from 491 medium-sized projects between 35,000 and 95,000 new or modified source lines of code. All were information systems completed in the last three years. "New or modified" limits the code metric to "Effective SLOC," that is, the work that was actually done. It excludes reusable components that require little or no new work.

The next step was to stratify the sample projects into five team-size groupings, as shown in Figure 1. There are three features of this stratification to note in particular:

- The number of projects in each group is substantial (so our conclusions can be credible);
- The data sets are fairly well distributed across the entire size regime;

- The average size of the stratified data sets is 57,412 Effective SLOC and the average of each set is within 3,000 ESLOC of this overall average.[fn]We are indebted to Douglas Putnam for making this analysis.[/fn]

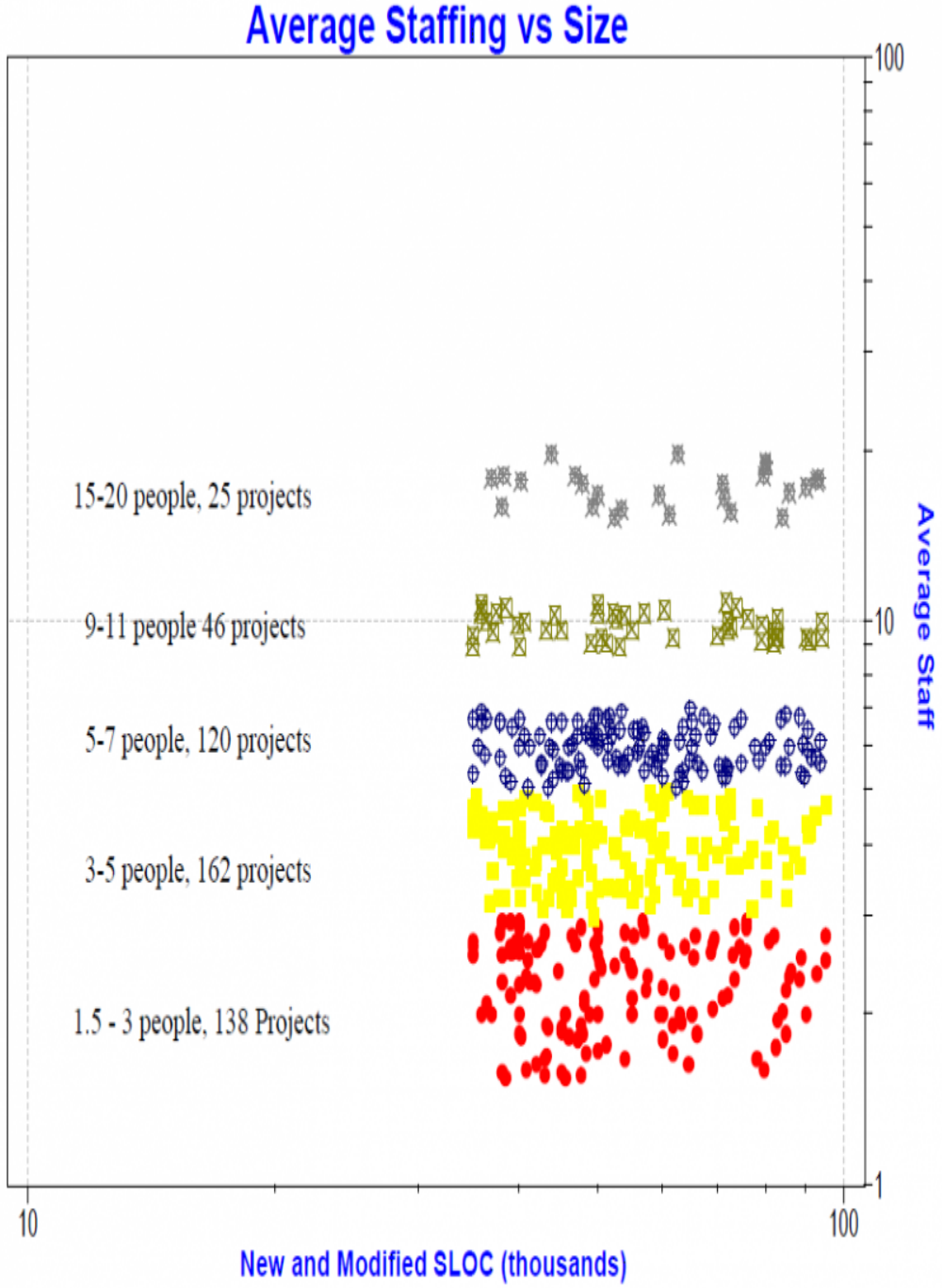

Figure 1. The projects are stratified in two dimensions: size horizontally and staff vertically. Since both axes are logarithmic, the diagram covers a large range of information.

The next figure shows that the three small groupings took less schedule time than the two large groupings (Figure 2). In fact, development time for the three small groupings is about three quarters of the time for the two large groupings.

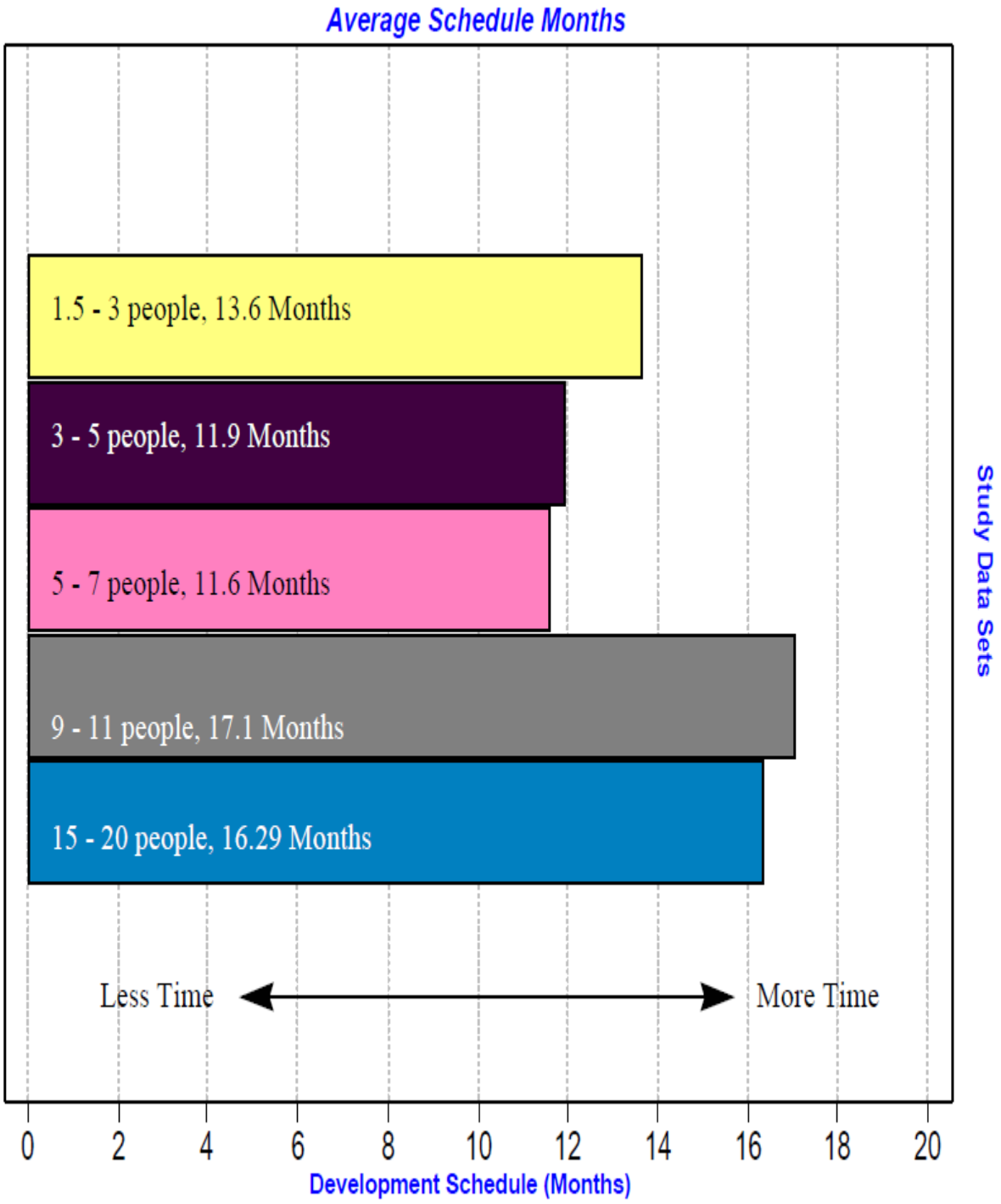

Figure 2. There is a distinct difference in the schedule time required by groups in the three-to-seven range as compared to the nine-to-20 range.

Small groups use fewer person-months

Figure 3 shows the pattern for development effort. The difference is much more marked than in the case of

schedule. The three small groupings take only about one fourth the effort of the two large ones—remember, each grouping is producing about the same amount of system functionality.

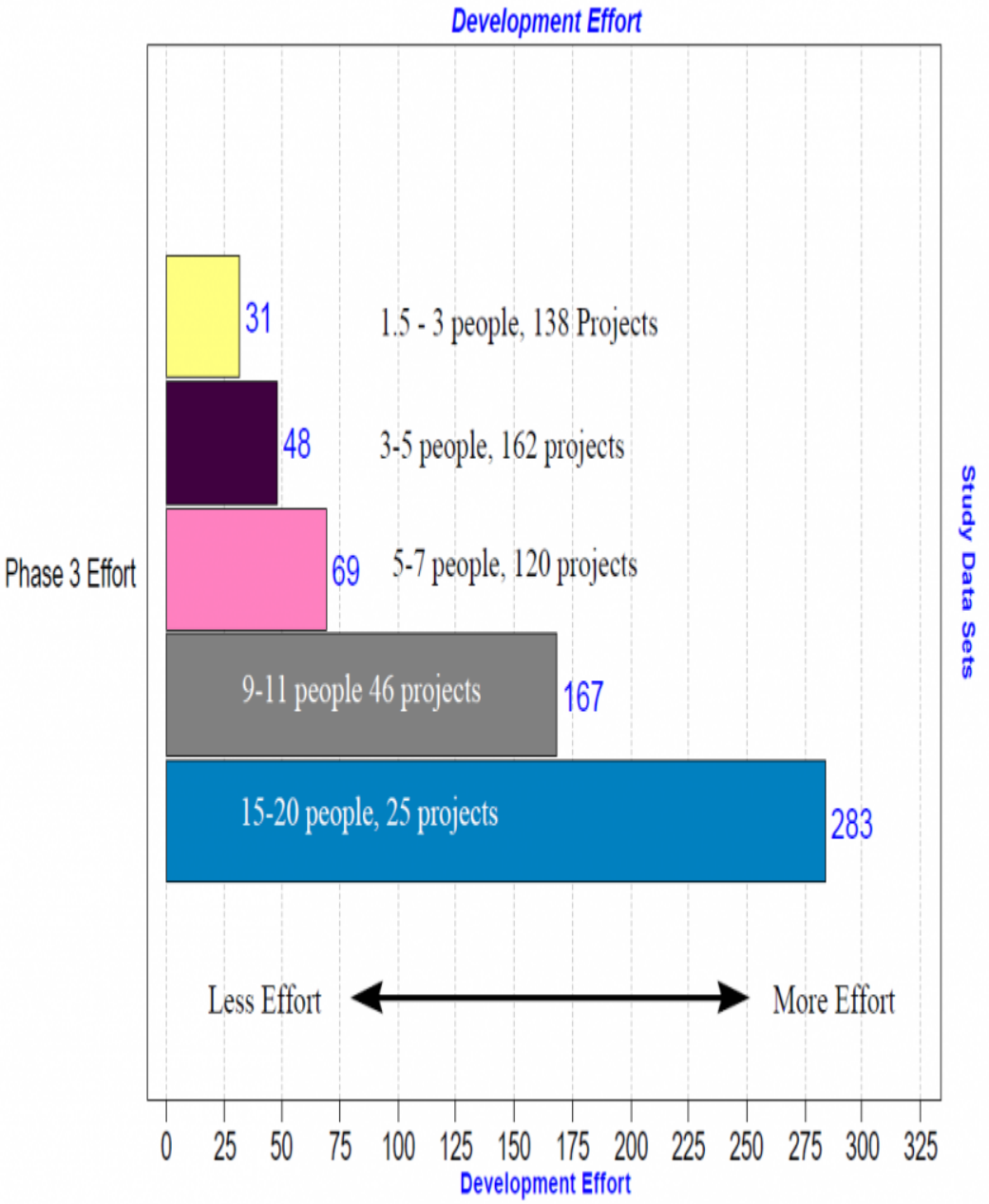

Figure 3. A distinct difference in the person-months needed to do a comparable amount of work begins to show up when group size exceeds eight people.

We present a somewhat different representation of the effort data in Figure 4. Here the projects completed by two-to-five person teams are located (on a log-log field) by small squares. The lower of the two slanting lines represents their average location. The projects completed by larger teams (seven to 14 people) are represented by the small circles and the upper line. At the 10 EKLOC size the larger team effort is about four times as great as the smaller team effort, and that ratio continues out to 100 K.

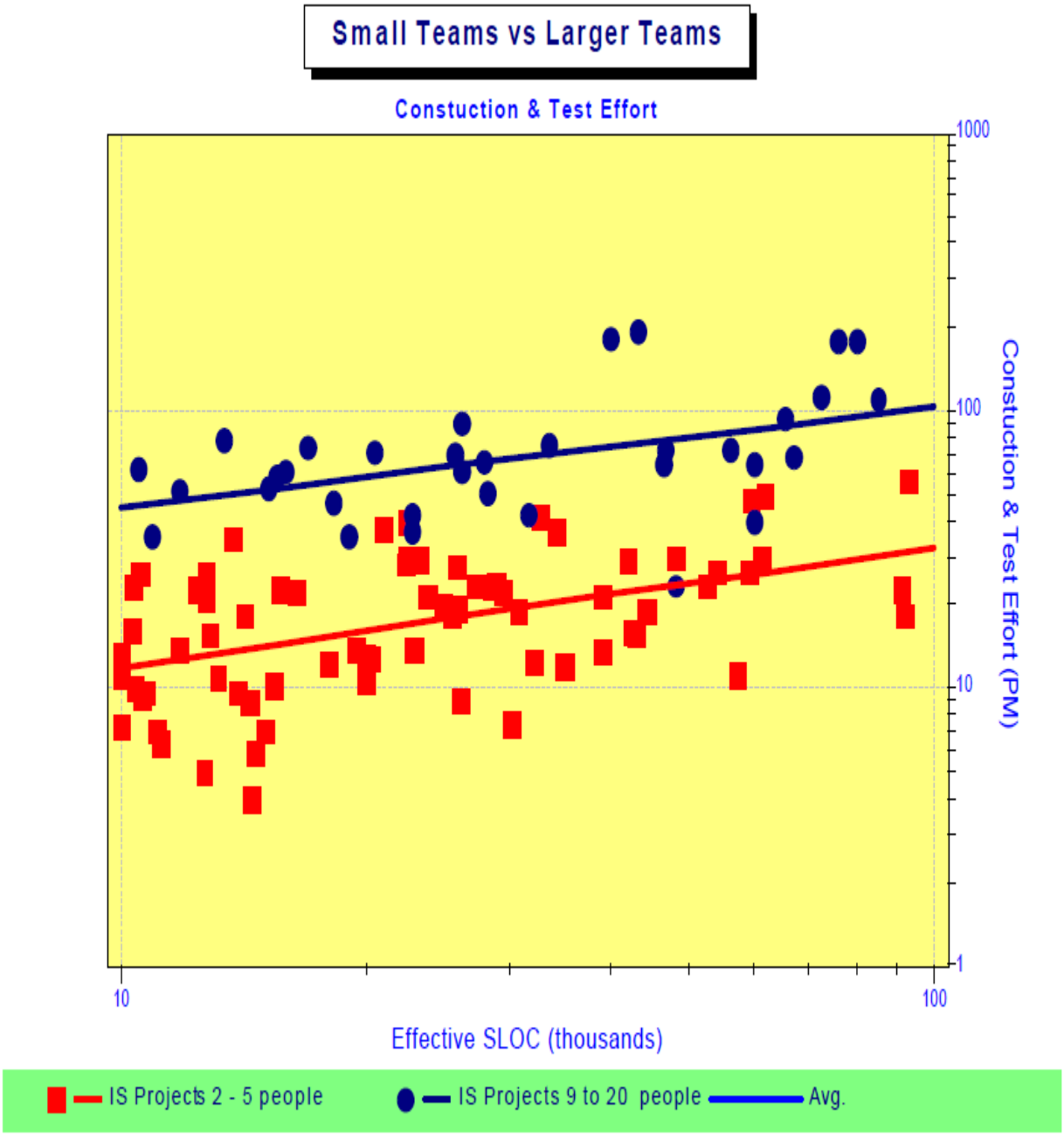

Figure 4. The larger teams (upper line and circles) take much more effort than the smaller teams (lower line and squares).

Process productivity reflects effect of group size

It comes as no surprise that the process productivity indexes of the three small groupings are about three index points higher than the indexes of the two large groupings, as Figure 5 details. The productivity indexes of the small groups average 16.28; those of the large groups, 13.38. That is a difference of approximately three index points. Each index point represents a gain of 1.27 times in process productivity over the previous index. A gain of three index points, therefore, doubles the process productivity $(1.27)^3$.

As you will recall from the software equation quoted at the beginning of this column, when project size is held constant, a few algebraic operations makes process productivity inversely proportional to the schedule and effort terms:

Process productivity = Constant / (Effort/B)$^{1/3}$ (Development Time)$^{4/3}$

So the equation says, when schedule and effort improve dramatically (that is, smaller numbers), process productivity improves significantly as well. And indeed the data show it did!

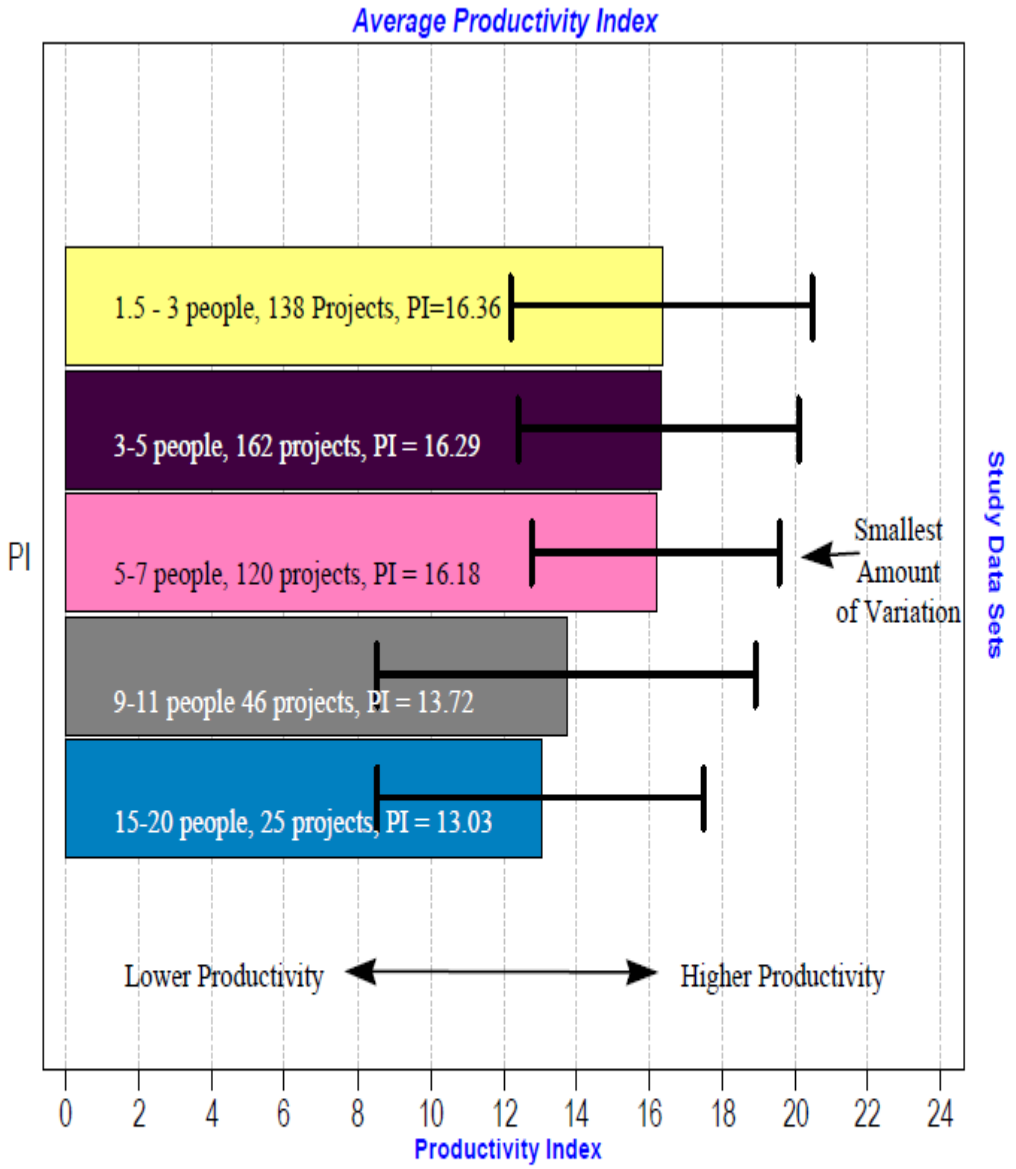

Figure 5. The process-productivity index is a linear expression of the actual process-productivity parameter, so the real difference in productivity (a factor of about two times) between the small groupings and the large groupings is muted on this linear diagram. The high-low variation bars are set at one standard deviation above and below the mean.

Optimum size is three to seven

Confirming the classical research of our son's textbook , our metrics verify that we should develop software in groups of three to seven. If we go to groups larger than seven, we pay a severe penalty in the time and effort a

project takes.

Of course, a “group” of one has many advantages. It can confer with itself sitting in a corner with its eyes closed. It need not spend time batting down not-so-brilliant ideas from other team members. But it has disadvantages: it goes on vacation, it gets sick, and it has no replacement. And so on.

A group of two has many pluses, too, as any happily betrothed man or woman can attest. Two is the number good old “evolution” arrived at. But the number is a bit unstable, as the number of orphans suggests. Still, don’t mind our negative attitude. Do use one or two folks on small suitable [non-strategic] projects.

On the great mass of mid-sized information systems projects (and comparable projects in other application areas) we are going all-out for three to seven.

What about really big projects?

The average project size, reported to us, has been declining from about 80 KSLOC in the early

1980s to about 40 KSLOC in the 1990s, as we diagrammed in our May 1998 column. Those sizes are well within the range of the present study. Still, there are occasional very large projects that we hear about when they fail. We agree: they cannot be done by a team of seven. Unfortunately, a project of 50 or 250 people is going to run right off our diagrams! What to do?

The immediate answer is obvious. Huge projects have to be split up into little projects that little teams can handle. You might have noticed that the last sentence has nobody doing the splitting up, and not by accident. We can nominate some titles: customers, project managers, architects, senior designers, stakeholders in general. Some of these people must know how to do it, because some very big jobs have been accomplished. In our experience, however, people with these abilities are very, very scarce. So, the not-so-immediate answer is shrouded in the fog of the future.