ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS

ESSAYS ON SOFTWARE ENGINEERING

# THE MYTHICAL MAN-MONTH

FREDERICK P. BROOKS, JR.

Photo credit: © Jerry Markatos

## ABOUT THE AUTHOR

Frederick P. Brooks, Jr., is Kenan Professor of Computer Science at the University of North Carolina at Chapel Hill. He is best known as the "father of the IBM System/360," having served as project manager for its development and later as manager of the Operating System/360 software project during its design phase. For this work he, Bob Evans, and Erich Bloch were awarded the National Medal of Technology in 1985. Earlier, he was an architect of the IBM Stretch and Harvest computers.

At Chapel Hill, Dr. Brooks founded the Department of Computer Science and chaired it from 1964 through 1984. He has served on the National Science Board and the Defense Science Board. His current teaching and research is in computer architecture, molecular graphics, and virtual environments.

# 8
# *Calling the Shot*

*Practice is the best of all instructors.*

*PUBLILIUS*

*Experience is a dear teacher, but fools will learn at no other.*

*POOR RICHARD'S ALMANAC*

Douglass Crockwell, "Ruth calls his shot," World Series, 1932

How long will a system programming job take? How much effort will be required? How does one estimate?

I have earlier suggested ratios that seem to apply to planning time, coding, component test, and system test. First, one must say that one does *not* estimate the entire task by estimating the coding portion only and then applying the ratios. The coding is only one-sixth or so of the problem, and errors in its estimate or in the ratios could lead to ridiculous results.

Second, one must say that data for building isolated small programs are not applicable to programming systems products. For a program averaging about 3200 words, for example, Sackman, Erikson, and Grant report an average code-plus-debug time of about 178 hours for a single programmer, a figure which would extrapolate to give an annual productivity of 35,800 statements per year. A program half that size took less than one-fourth as long, and extrapolated productivity is almost 80,000 statements per year.[1] Planning, documentation, testing, system integration, and training times must be added. The linear extrapolation of such sprint figures is meaningless. Extrapolation of times for the hundred-yard dash shows that a man can run a mile in under three minutes.

Before dismissing them, however, let us note that these numbers, although not for strictly comparable problems, suggest that effort goes as a power of size *even* when no communication is involved except that of a man with his memories.

Figure 8.1 tells the sad story. It illustrates results reported from a study done by Nanus and Farr[2] at System Development Corporation. This shows an exponent of 1.5; that is,

$$\text{effort} = (\text{constant}) \times (\text{number of instructions})^{1.5}.$$

Another SDC study reported by Weinwurm[3] also shows an exponent near 1.5.

A few studies on programmer productivity have been made, and several estimating techniques have been proposed. Morin has prepared a survey of the published data.[4] Here I shall give only a few items that seem especially illuminating.

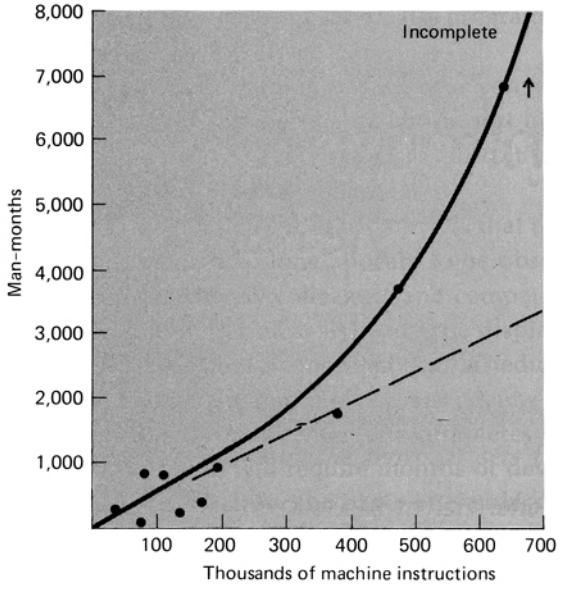


**Fig. 8.1** Programming effort as a function of program size

## Portman's Data

Charles Portman, manager of ICL's Software Division, Computer Equipment Organization (Northwest) at Manchester, offers another useful personal insight.[5]

He found his programming teams missing schedules by about one-half—each job was taking approximately twice as long as estimated. The estimates were very careful, done by experienced teams estimating man-hours for several hundred subtasks on a PERT chart. When the slippage pattern appeared, he asked them to keep careful daily logs of time usage. These showed that the estimating error could be entirely accounted for by the fact that his teams were only realizing 50 percent of the working week as actual programming and debugging time. Machine downtime, higher-priority short unrelated jobs, meetings, paperwork, com-

pany business, sickness, personal time, etc. accounted for the rest. In short, the estimates made an unrealistic assumption about the number of technical work hours per man-year. My own experience quite confirms his conclusion.[6]

## Aron's Data

Joel Aron, manager of Systems Technology at IBM in Gaithersburg, Maryland, has studied programmer productivity when working on nine large systems (briefly, *large* means more than 25 programmers and 30,000 deliverable instructions).[7] He divides such systems according to interactions among programmers (and system parts) and finds productivities as follows:

| | |
|---|---|
| Very few interactions | 10,000 instructions per man-year |
| Some interactions | 5,000 |
| Many interactions | 1,500 |

The man-years do not include support and system test activities, only design and programming. When these figures are diluted by a factor of two to cover system test, they closely match Harr's data.

## Harr's Data

John Harr, manager of programming for the Bell Telephone Laboratories' Electronic Switching System, reported his and others' experience in a paper at the 1969 Spring Joint Computer Conference.[8] These data are shown in Figs. 8.2, 8.3, and 8.4.

Of these, Fig. 8.2 is the most detailed and the most useful. The first two jobs are basically control programs; the second two are basically language translators. Productivity is stated in terms of debugged words per man-year. This includes programming, component test, and system test. It is not clear how much of the planning effort, or effort in machine support, writing, and the like, is included.

| | Prog. units | Number of programmers | Years | Man-years | Program words | Words/man-yr. |
|---|---|---|---|---|---|---|
| Operational | 50 | 83 | 4 | 101 | 52,000 | 515 |
| Maintenance | 36 | 60 | 4 | 81 | 51,000 | 630 |
| Compiler | 13 | 9 | 2¼ | 17 | 38,000 | 2230 |
| Translator (Data assembler) | 15 | 13 | 2½ | 11 | 25,000 | 2270 |

**Fig. 8.2** Summary of four No. 1 ESS program jobs

The productivities likewise fall into two classifications; those for control programs are about 600 words per man-year; those for translators are about 2200 words per man-year. Note that all four programs are of similar size—the variation is in size of the work groups, length of time, and number of modules. Which is cause and which is effect? Did the control programs require more people because they were more complicated? Or did they require more modules and more man-months because they were assigned more people? Did they take longer because of the greater complexity, or because more people were assigned? One can't be sure. The control programs were surely more complex. These uncertainties aside, the numbers describe the real productivities achieved on a large system, using present-day programming techniques. As such they are a real contribution.

Figures 8.3 and 8.4 show some interesting data on programming and debugging rates as compared to predicted rates.

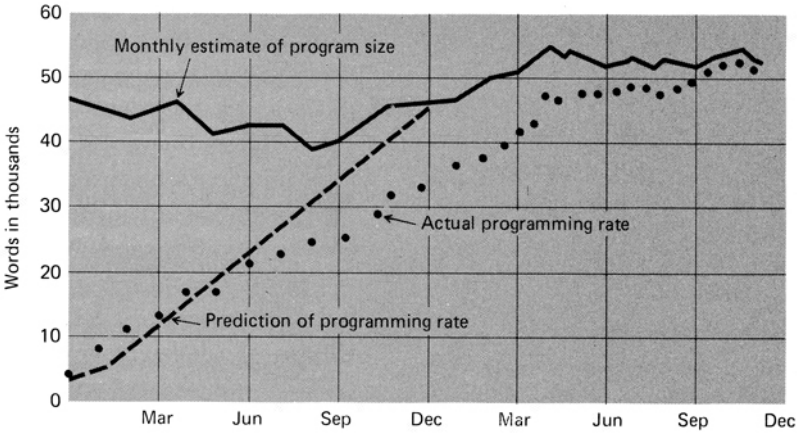


Fig. 8.3 ESS predicted and actual programming rates

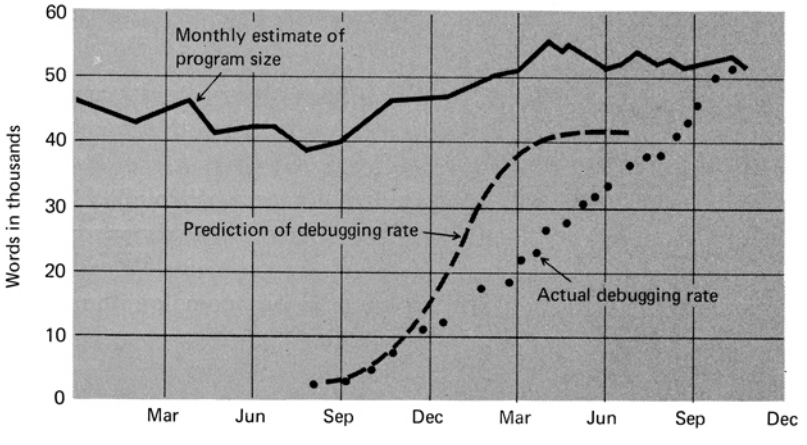


Fig. 8.4 ESS predicted and actual debugging rates