NOTES ON ESTIMATING AND OTHER SCIENCE FICTION

By Carl H. Reynolds

President, Computer Usage Development Corp.

Guidelines for estimating the costs of programming have always been somewhat of a mystery due to the large number of variable factors affecting programming. Some of these factors are discussed here.

Systems Development Corporation published two reports recently which bear on one of the most difficult aspects of programming, namely, estimating the cost to produce a new program. The first is a handbook for management for programming estimating; the second is a little report which is the result of a small study to attempt to determine the effectiveness of time sharing computers in reducing programming time.

The management handbook on estimating is the result of several years' work of research into the factors which affect the cost of systems definition, design, development, and maintenance. The handbook is interesting for a couple of

reasons. First, in the areas of definition and design, it lists all the factors which can affect the cost. I would guess there are over 150 different attributes of a problem which are listed with their potential impact.

It amounts to a delineation of common sense and is, in fact, a standard example—based on a certain amount of measurement—of what we use at Computer Usage to try to define the uncertainty in our estimating. The things we cover, and which are covered in the report in much greater detail, are: familiarity of personnel with the machine to be used; familiarity with the programming system; familiarity with the application; reliability of the machine; reliability of the programming systems; characteristics of the agency for whom the program is to be done (i.e., they are cooperative and knowledgeable, all the way down to uncooperative and unknowledgeable); and the physical environment surrounding the systems design. For example, are people near home or do they have to travel extensively.

In brief, the conclusion on this part of the report is that there are many factors which can increase or affect the certainty with which an estimate can be developed. It is significant that after all the study, all that SDC can do is indicate the relative influence of all the factors. That is, it shows which are most significant and which are least, but cannot give very many concrete guides as to how to apply them.

Costing the Implementation

The second part of the report deals with estimating of actual coding and debugging. The approach taken here was to gather as much data as possible about as many specific systems implementations as possible and then attempt statistically to correlate the known costs with possible factors such as language used, number of debugged instructions produced, size of machine, and so forth. The result is a set of equations which might be used to predict the cost of implementing a specific design. I say might, because in testing these equations against the original statistical data the results are less than conclusive.

The report closes with a plea that more and more detailed information be gathered about those factors which might affect cost, so that in the future things could be predicted better.

The second report is on a small experiment designed to test the assumption that on-line time sharing terminal systems will, in fact, significantly reduce programming time and cost. While the results do indicate a significant but relatively small decrease in programming costs, the most striking thing is that the cost of programming is many times more sensitive to the skill of the programmer than it is either to a batch or time shared mode of programming.

Affected By Three Factors

Now, I read into these two reports some old convictions. First, one cannot expect an accurate estimate of a brand new task on brand new machines even if the people are experienced. Two, one cannot expect an accurate estimate on an old application on well-established hard and software with inexperienced people. Three, the biggest single variable in estimating is the quality of personnel applied to it.

What can we as managers do, then, to assure not only good estimates but quality, low-cost production? The obvious thing to do is to have only outstanding programmers. Barring that—sometimes it's hard for us even to recognize and find them when we have them, much less keep them motivated—it seems to me the most important part of this job is to force the definition of programming tasks prior to their start.

We only have a few things available to us which make easy work definition. The first, of course, is specifications. Second is systems design. It is distasteful to many professionals, but about the only way the system design can be observed at the current state of the art is through a flowchart. Finally, of course, there is coding, starting of testing and test completion.

My contention is that if the work is laid out with checkpoints at the completion of at least the first draft of each of these items, and if records are kept of the time and energy and machine time taken to achieve these steps, we will gradually develop rules of thumb which apply at least within the application and the machine environment in which we operate as individuals. It takes a lot of energy to insist upon the establishment of checkpoints before work begins and accurately and completely maintain them during work, particularly under the pressure of schedule slippages and cost increases. This, however, is the only way in which we can keep growing in our knowledge of the activity.

Objective Evaluation Needed

One caution, though. The only man who can really make an estimate is the one who has done it and will have to do the work. In addition there grows up a professional commitment, not to the job, but to the estimate, and programmers often have a feeling that if they change the estimate they have somehow failed. This kind of feeling—often generated by management's unwillingness to ferret out the real causes for schedule problems, and their own desire not to look stupid in the eyes of their own management—tends to cloud the issue and keep from both management and the programmer an objective recognition of where the work really stands.

An estimate obviously can never be expected to be 100 percent accurate, even if it is to do precisely the same job over again. Almost never do we have that luxury in estimating. Thus it behooves management to take a constructive, flexible attitude toward the schedule once it is developed. The problem of motivating people to complete the job in the least possible time is a task too often left to an inflexible schedule commitment, rather than to the other techniques of management.

Rigid achievable deadlines are indeed indispensible to making any kind of schedule, whether it's building a building or a program. There exists, however, real problems which are unanticipated which are better faced directly than just avoided because of management fear.