Managing Application Programming With Fourth Generation Resources

N. M. Demos
Performance Software Group
Baltimore, Maryland 21228

In the late fifties IBM realized that its computer effort was fragmented into several different computer models employing several different architectures. At the same time, system software, particularly operating systems, were recognized as an integral part of a computer vendor's offering. To meet this challenge, IBM after much soul searching and internecine warfare, announced the System/360. While electronically different and designed and produced by different engineering groups and plants, each model would have the same instruction set and peripheral interfaces. Therefore programs could be ported at the object code level from one model to another. This would make upgrading much easier for the user and optimize programmer resources, because only one instruction set and architecture would have to be learned.

Software could be designed and written based on the same instruction set. This was a critical element in the strategy and success of the S/300 architecture.

The other radically new feature of the S/360 is that it was designed to be used with an operating system. The operating system, using the new capabilities of the disk drive, would facilitate program to program transition, handle all input and output, and allow multi-programming. IBM initially announced two operating systems for the S/360 - DOS and OS (there was also initially a tape varient of DOS called TOS). OS, designed for the larger and faster S/360 models, was designed to have a functionality way beyond the capability of anything seen previously in a commercially available computer system.

Brooks' "The Mythical Man Month" is an analysis of what happens when a very large system design and programming effort - in this case IBM's OS - is undertaken. While I recommend that everyone read this book, as its lessons are still germaine today and many of its precepts ignored, I can tell you what the message I received from the book was.

Beyond the obvious - man and machine resources - the book has some very important insights on how to bring a large system to completion on time and within budget. As well as showing some of the human frailties that make a project manager's life so difficult, it emphasized that only by superior organization and a thorough understanding of the task could a large project be accomplished. Brooks estimates that a program that interfaces to other programs takes at least three times as much time.(1) He also states that "The man month as a unit for measuring the size of a job is a dangerous and deceptive myth."(2) First, this confuses effort with accomplishment, which are not the same. More important, this type of scheduling assumes that a job "can be partitioned among workers, with no communication among them."(3)

There are two tasks that have to be carefully accomplished for the project to be successful. In the first place, the project must be logically organized

and then broken down so that each person has a task that can be understood and accomplished by one person. At the same time all interfaces, vertically and horizontally, must be thoroughly, absolutely and immutably defined. Of course, we all know that that is impossible, but we must get as close as possible. Incidentally, it would seem to be in this area that the Spectrum project may have gone astray.

Once having designed and written specifications, the project must be organized and scheduled. Brooks sees another pitfall here. Ideally, one would like to have a "small sharp team" - a small group that knows the tasks thoroughly, communicates with each other effectively and can perform effectively together. Unfortunately, this group of near geniuses is mostly unavailable and in any case, cannot handle large tasks in a timely manner. Brooks proposes instead project teams of specialists with only one person taking responsibility for all the code, most of which he would write himself.

How have some of the more recently available resources, particularly Fourth Generation languages affected the project director's ability to perform these tasks? First of all, a good fourth GL, because code can be written so much faster and modified more easily, allows realistic prototyping, so that both the user and designer can quickly determine if they are on the right track. This not only verifies the design, but gets the user involved and therefore he is more likely to become committed to the project. Today, with most user interaction occurring via a display, it becomes critical that the users understand what they are expected to do to accomplish the application's objectives. This means that not only must they become comfortable with the terminal, but it must be easy and as obvious as possible to enter data and perform other functions correctly. Prototyping is one way of making this happen. In many cases the last prototype becomes the production version. In other cases, the prototype is a partial solution to the task and is used until the complete system is available. In situations where the prototype or a portion of it will be used in production, the fourth GL and the user implementation must be robust enough to operate in a production environment.

A good fourth GL supports a dictionary. This is a major asset in standardizing data definitions, linkages, and databases so that all programmers access the same data structures in their programs. If carefully managed, not only does this help implement standards and prevent errors caused by programmers not having up-to-date specs, but it facilitates changes. A good dictionary can be used for even more than that. Depending on the dictionary and the skill of the dictionary user, it can store in accessible form definitions on all linkages and program code, thereby allowing members of the project team to query it for as complete a map of the project as they might require.

When a fourth GL is chosen, each one under consideration must be analyzed not only for what it can do itself, but for the environment it operates in. For example, if documentation is an issue, then Infocentre's Documentor supplies a solution not available with the other fourth GL's. If the applications are complex in logic, then the fourth GL must have the capability to accept procedural code and run it on the 3000 without performance degradation. For

example, the combination of HP's Transact and Performance Software Group's FASTRAN accomplishes this objective.

The bottom line on project planning is still the same - the better the planning and organization, the better chances for a successful implementation. The availability of new facilities such as 4GL's and dictionaries, if properly utilized, give the project management powerful new tools to help him.