and two had no previous experience with PL/I. All training in both the design techniques and in PL/I programming are included in the project.

As mentioned previously, our new SP techniques emphasize the design phase. In the design phase we used a top-down approach, and we used the structured coding constructs in our PDL programs. In addition, we were able to iterate the design many times because the investment in a procedure in PDL is much smaller than in the same procedure in code. If someone could see a better approach to a problem, then it was redone with the new approach. If two parts of the design looked sufficiently similar to warrant the introduction of a common utility routine, then that was done and the designs of the callers were changed to reference the new utility. Such things were possible because the design was precise and also aimed at people. When we finally started coding, it was as if we were rewriting an already existing program.

During the actual implementation phase we used the structured coding and top-down implementation techniques as before. Since we had great confidence in what we were coding, we coded large sections before testing and, therefore, avoided completely the use of stubs. We did build a driver to test our common utilities before we integrated them with the remainder of the program.

We placed extensive trace facilities in the program as it was coded and debugged by turning on the proper parts of the trace. Since we had built a project-oriented dump facility, we always worked in the batch mode and never obtained regular OS/360 dumps. We worked as a team both in the design and in the implementation phases. The team approach is especially important in the design phase because the only verification of the correctness of the work is by human understanding.

Conclusions

We believe our new methods gave us more improvement over our previous attempts at SP than those earlier methods gave over conventional methods. This showed up most in the quality of the code and in the amount of machine time we used during the project. Our increase in lines per day was not so large, but we believe we solved the problem with many fewer lines of code by these methods.

Concluding Remarks

Barry W. Boehm
TRW Systems

At this point, there is no good reason for an organization not to take steps toward adopting some variant of SP techniques. The particular variant will depend a lot on your organization: factors such as the type of applications, personnel, languages used, inventory of existing code, organizational constraints, and typical project size can lead to different preferred approaches. A careful planning and training phase should precede use of the techniques, in order to minimize the risks of their misuse. With this preparation, the benefits of using a suitably tailored SP variant will significantly outweigh the risks.

At this point, there is no way to predict the magnitude of the benefits your installation will derive from using SP. It won't be long before you hear the 40 to 50 percent savings figures in Table 1 quoted out of context to support all sorts of sales pitches. Here's an example of a recent one, from a highly reputable organization, pitched toward a typical medium-sized programming shop with 8 people doing program development and 12 doing program maintenance:

“If you buy our Structured Productivity Software Support Package, here’s what will happen. Since SP makes maintenance 50 percent more productive, you’ll be able to do your maintenance with just 8 people, leaving 12 for development. And since SP makes development 50 percent more productive, you’ll actually be able to do 18 people’s worth of development work each year instead of only 8.”

Aside from the fallacious logic (how will the 8 maintenance guys handle the higher volume of newly developed software?), the assumption that your shop will boost its productivity by 50 percent is totally unwarranted. Caveat emptor.

The most important thing of all to keep in mind is that increased programmer productivity is not an end in itself. It is simply another means to better serve data processing users. Thus, all the quantitative data given here on instructions produced per man-month should really be considered subordinate to the qualitative information provided in the descriptions above on the effect of using SP on programmers’ attitudes and on the degree of user satisfaction with the product.

Unfortunately, the current trend in SP approaches is far too heavily oriented toward programming productivity and far too little concerned with the user. An example is the
prevalent answer to the question, "Where's the top?" in top-down development. Here are the kinds of answers most commonly encountered in SP articles, courses, and presentations today:

"The real top is the operating system itself."
"Job Control Language."
"The data definitions and overall control logic."

Here are typical examples given to students, programmers, analysts, and managers as representative of the "top" of a top-down development:

"Read a sales-slip file and produce a file of bills."
"Print the first 1000 prime numbers."
"Add a member to a library of program elements."
"Monitor a patient's vital factors."

This last example is the only one of the above that defines the "top" in terms of user considerations. The others tend to channel one's thoughts exclusively toward programming considerations.

If there is anything disquieting about the SP movement today, it is the future prospect of legions of programmers and computer science graduates soaking up the above examples and then having to be painfully retrained as to where the real top is in information system development.

The real top is the user. SP practitioners who forget this (or are taught to think otherwise) may produce code very efficiently, but if they have to rewrite 95 percent of it to meet the user's needs (as has happened in practice), their effective productivity will be pretty dismal. On the other hand, user-oriented SP practitioners will find the top-down approach useful in achieving responsive as well as efficient software development.

References


Barry W. Boehm is Director of Software Research and Technology of the Systems Engineering and Integration Division of TRW Systems. He entered the computing field as a programmer in 1955, and has served variously as numerical analyst, system analyst, information system development project leader, manager of groups performing such tasks, head of the Information Sciences Department at The Rand Corporation, and study director of the Planning, Programming, and Budgeting System for the Department of Defense. He is the author of papers on a number of computing subjects, most recently in the area of software requirements analysis and design technology. He recently directed a study for the U.S. Air Force on the configuration management implications of structured programming and other new software technologies.

Dr. Boehm received his BA in mathematics from Harvard in 1957 and his MA and PhD from UCLA in 1961 and 1964, respectively.

Charles E. Holmes is the manager of Business Systems Services at McDonnell Douglas Automation Company, St. Louis, Missouri. Starting as a scientific programmer in 1956 at McDonnell Aircraft Corp., he has held various supervisory and management positions within computer operations, and scientific and business programming. He received the BS degree in mathematics from St. Louis University in 1951.

Gene R. Katkus is a staff engineer on the Advanced Programs Staff of the Computer Programming Laboratory at Hughes. In this position, he is responsible for initial software definition, top-level software architecture design, performance specification, sizing, software development planning, etc., for proposed software systems. Previously, he was currently software technical director for the Contract Definition Phase of both the SURveillance Towed Array Sonar Segment (SURTASS) and the Escort Towed Array System (ETAS) and was the chief programmer on Project HUSAR.

He received the BS degree from the University of Illinois in 1966, and the MS and PhD degrees from the University of Southern California in 1968 and 1973, respectively, all in electrical engineering. He was a recipient of Hughes Masters and Doctoral Fellowships and is a member of Eta Kappa Nu, the ACM and IEEE.

James P. Romanos is presently employed as an engineering specialist on staff to the manager of the Software Engineering Department of GTE-Sylvania in Mountain View, California. He is currently leading a software design effort using structured design techniques, for a large real-time command and control system. Previously he was employed by Hughes Aircraft Co. and by General Electric Corp. Romanos received his BA in mathematics from the University of Connecticut. He is a member of the ACM and IEEE.

Robert C. McHenry is the manager of TRIDENT CCS Integration Software, IBM Federal Systems Division. In his previous assignment he was manager of programming technology. He has managed several government and commercial projects. Prior to joining IBM in 1959 he was with the National Security Agency for eight years.

E. Kent Gordon is a vice president and founder of Caine, Farber & Gordon, Inc. He is responsible for the company's large scale operating system developments and for its activities in the area of modern programming technology. He was previously Manager of Batch Processing Systems at the California Institute of Technology.