Concurrent phasing: When time means money

by RICHARD G. LEFKON
Citibank, N.A., and New York University
New York, New York

INTRODUCTION

Those responsible for improving the efficiency of software generation and maintenance, sometimes overlook an obvious but nonetheless critical fact: If there were no users, there would be considerably fewer programmers to manage.

One problem with large-scale programming efforts is that by the time the system is finished, the business it serves may have evolved into something distinctly different. Users will not stand by quietly without good cause if their key systems are frozen for a lengthy period to accommodate the programming department.

EXTRA COSTS MEAN SAVINGS!

When time means money, time frame acceleration can make a large-scale project more costly, but the extra expense may be more than justified by a much larger return on the investment. Bringing a new system live significantly earlier means that the resulting benefits in savings, profit increment or marketplace leadership are cumulatively in effect for that much longer. Business users pay attention to this return on investment.

Even from the programming department’s standpoint, there are benefits:

Employee salaries and consultant costs will be billed at today’s rates rather than increase steadily over time; the new system can use state-of-the-art software rather than, for instance, a database package that has aged several additional years upon system delivery; the necessary freezing of present procedures will be shorter and cause correspondingly less disruption to the conduct of business.

INTRODUCTION TO CONCURRENT PHASING

Concurrent phasing is a two-dimensional project management approach that integrates the management practices of task subdivision, subtask ganging, functionally distinct development teams, early completion of the common database, and extensive prototyping. This approach facilitates coordinated delivery of subsystems to the user as integral wholes. It makes testing, fixes, and user sign-off much less awkward.

The chief benefit of concurrent phasing is that it smooths and reduces the staff load required to complete a software development project in a greatly accelerated time frame. The final software products delivered are not necessarily any better than those produced with other approaches, and completing a project in 1½ years with this strategy will cost significantly more than permitting the same project to take five years to go live. But when time means money in its effect on the business, this more expensive project management strategy may be just the right one.

HOW TO IMPLEMENT CONCURRENT PHASING

Concurrent phasing is definitely not the place for laissez-faire managers. Because many tasks usually done in sequence will now be performed concurrently, the project management team must take an active role, meeting at least weekly to discuss problems and alternative solutions.

Senior management may be unfamiliar with concurrent phasing and at first may have difficulty conceptualizing this management-intensive method. It is imperative that this accelerated approach be presold to those with systems influence in either the data processing or end-user departments and explained carefully as new individuals enter the management chain. Because it runs counter to the traditional linear schedule, concurrent phasing should periodically be discussed with those to whom it has already been explained. A loud “This method will never work!” can decelerate progress if left unanswered often enough.

A simple memorandum format can be used to express the worth of the management processes involved here. Such a memo, circulated in one’s own department and subsequently discussed with key users, can offer the following recommendations:

1. A prerequisite starting point is a complete set of known data items.

2. The first internal product delivered should be a comprehensive database—thinel populated but logically complete.

3. Before the conclusion of formal specifications, coders should construct a skeleton of the full system for tuning and hands-on feasibility studies.
4. Major subsystems should be planned, designed, and coded and tested with phased starting points. This will leave intact the three respective teams throughout the project, guarantee team-to-team handoffs, and minimize misuse of staff such as coders interviewing users, CICS learners designing transmissions, or user contacts writing code.

5. The management and staff of each team should work together to report progress based on the smallest components of each task and to gang labor on critical-path items.

6. To facilitate phasing, a list of cumulative features for subsystems should be enclosed.

LIMITATIONS

Except for subtasking, the synthesis of techniques discussed here will not produce the projected acceleration with projects of less than four to six work-years. Not only do projects of shorter duration have insufficient resources to facilitate ganging; but also the complexity of the task is probably not great enough to reap a net time gain from prototyping.

Software development projects managed by concurrent phasing still need the application of other accepted good programming management practices. Coding standards must be introduced and enforced, and quality testing must be thoroughly and consistently applied. Moreover, non-programming considerations affecting users cannot safely be ignored either.

Even within this approach, common sense should be applied. Coders should be discouraged from breaking up a program in circumstances where the time for component linkage coding and testing is comparable to the time to be gained by coding in parallel (ganging). And, too, subordinates must understand that the deadlines are real and that 80 percent completion is not good enough.

Finally, if a new software technology is used, veteran practitioners of only the old technology should be kept off the project design team. Otherwise, their knowledge of the business, seniority, and self-confidence may converge so strongly that they lead the real software experts to come up with the wrong system architecture.

REFERENCES

3. Leikon, R. G. "Maintenance manager: How to be a drill sergeant and a good guy, too." Computerworld, 21(February 9, 1987)5, pp. 61–75.

<table>
<thead>
<tr>
<th>Responsible Individual or Group</th>
<th>Period I</th>
<th>Period II</th>
<th>Period III</th>
<th>Period IV</th>
<th>Period V</th>
<th>Period VI</th>
<th>Period VII</th>
<th>Period VIII</th>
<th>Period IX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Analyst:</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>train operators and clerical staff ..........</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Systems Analyst:</td>
<td>Design specifications</td>
<td>A specifications</td>
<td>B specifications</td>
<td>C specifications</td>
<td>D specifications</td>
<td>E specifications</td>
<td>perform maintenance and small upgrades</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programmer/Analyst:</td>
<td>Create a model system</td>
<td>Create the database</td>
<td>A unit testing unit testing</td>
<td>B unit testing unit testing</td>
<td>C unit testing unit testing</td>
<td>D unit testing unit testing</td>
<td>Perform maintenance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control:</td>
<td>Procedures</td>
<td>Document</td>
<td>Ensure Database</td>
<td>A integration testing</td>
<td>B integration testing</td>
<td>C integration testing</td>
<td>D integration testing</td>
<td>E integration Maintenance testing</td>
<td></td>
</tr>
<tr>
<td>User:</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D, E</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
</tbody>
</table>

TABLE I—Five subsystems with phasing and balance loading

From the collection of the Computer History Museum (www.computerhistory.org)