# Planning and Executing Time-Bound Projects

**The SPID approach combines critical chain planning with incremental development and rate monitoring to help software developers meet project deadlines.**

*Eduardo Miranda*
Ericsson Research Canada

A time-bound project is constrained by hard deadlines in which the timing of the delivery is as important as the delivery itself. Another way to put it is that if you deliver after the deadline, the delivery loses much of its value. Examples of hard deadlines include exhibition dates, competitors' announcements, and government-imposed regulations.

As with many other projects, most time-bound projects start with more requirements than developers can realistically handle within the imposed time constraints. As a result, they often have to start slashing these requirements halfway through the project, resulting in missed deadlines, customer frustration, and wasted effort.

A better approach is to define requirement priorities prior to starting a project and then allocate their development to successive releases of the project. With this approach, ever under severe adversity, the development team can guarantee delivery of the most important requirements by the deadline while still having a fair chance of completing less important requirements.

But failing to prioritize requirements is not the only reason that projects miss deadlines. The inability of traditional planning methods to deal with the uncertainty of estimates on which the plans are based and the failure to recognize that development work does not progress in linear fashion (the infamous 90-percent-complete syndrome) are also to blame.

Traditional critical-path calculations that attempt to address development uncertainties tend to produce considerably shorter schedules than what is realistic. With a shorter schedule as a starting point, being late often occurs almost by definition. The second problem, assuming that a task progresses at a constant rate, prevents project managers from seeing the early signs of delay until it is too late to take any other action except to trim features, compromise on quality, or reschedule the project.

Statistically Planned Incremental Deliveries (SPID) addresses these problems by combining ideas from critical chain planning,[1,2] incremental development,[3] and rate monitoring[4] into a practical method for planning and executing time-bound projects. SPID focuses on how best to organize a project to guarantee delivery of at least a working product with an agreed subset of the total functionality by the required date.

## TASK UNCERTAINTIES

Uncertainty is the root of all evil. If there were no variability, the solution would be simple. We could just make good plans and execute them as planned. The problem is that no matter how good the plans are, the estimates on which we base schedules and allocate resources rest on hundreds of assumptions about the complexity of the tasks, the ability of the development team to execute those tasks, the ability of the suppliers to deliver on time, the availability of a certain technology, the stability of the requirements, and even assumptions about the things we don't know.

If these assumptions turn out to be valid, some will contribute to the early completion of a task while others will add to its execution time. To complete a task at the earliest possible time, all the favorable assumptions must be true and all the

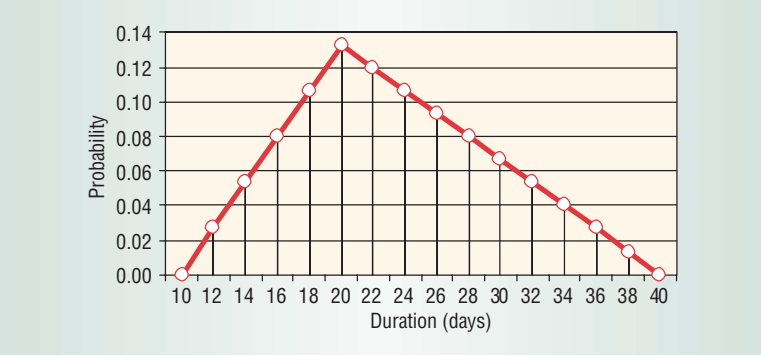


***Figure 1. Triangular probability distribution. If all the favorable assumptions turn out to be true and all unfavorable assumptions turn out to be false, the team will complete the task in 10 days—the earliest completion date. The most likely completion date, however, is 20 days. If everything that can go wrong does go wrong, the team will complete the task in 40 days—the latest completion date.***

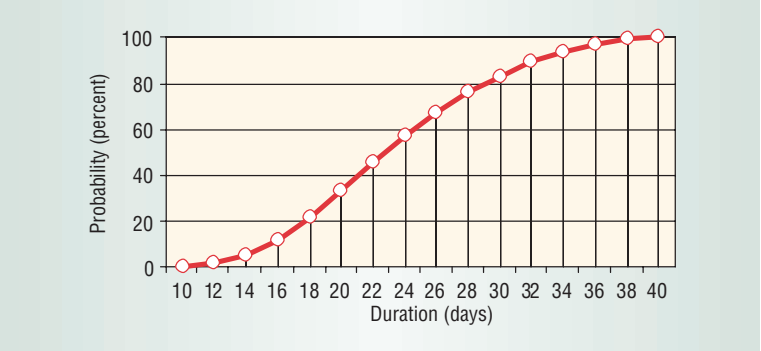


***Figure 2. Cumulative distribution. The most likely completion date has an on-time probability of less than 40 percent. In this case, the expected completion date is around 23 days. To be 75 percent sure of completing the task on time, the schedule must allow 27 days.***

unfavorable assumptions must be false. The probability of this happening is very low. We can say the same thing about the latest possible completion date. The most likely date corresponds to a situation in which the most probable favorable assumptions are true and the most probable unfavorable assumptions are false. The triangular probability distribution shown in Figure 1 expresses these observations numerically.

Since the actual probability distribution function for the duration of the task is unknown, a simple triangular distribution is a sensible choice.[5] A right-skewed triangular distribution captures the idea that while you can do only a limited number of things to shorten the duration of a task, the number of things that can go wrong is almost unlimited.

From the project-management point of view, the probability of completing a specific task on or before a certain date is more important than the probability of finishing the task on a specific date. The cumulative distribution shown in Figure 2 determines the task's on-time probability.

In general, the greater the number of assumptions behind a task's duration, the larger the spread between the earliest and latest completion dates. Such uncertainty results in very different on-time probabilities, as Figure 3 shows.

## FROM TASK TO PROJECTS

In moving from tasks to projects, conventional critical path planning relies on using the central limit theorem to assess the project's uncertainty. According to this theorem, the distribution of the sum of several independent random variables approaches a normal distribution as the number of variables grows larger. To assess project uncertainty using this approach, you calculate the expected duration of the project as the sum of the expected task duration along the critical path, with a standard deviation equal to the square root of the sum of the squares of the standard deviation of the same tasks. Then you use a normal distribution to calculate the on-time probability for the project.

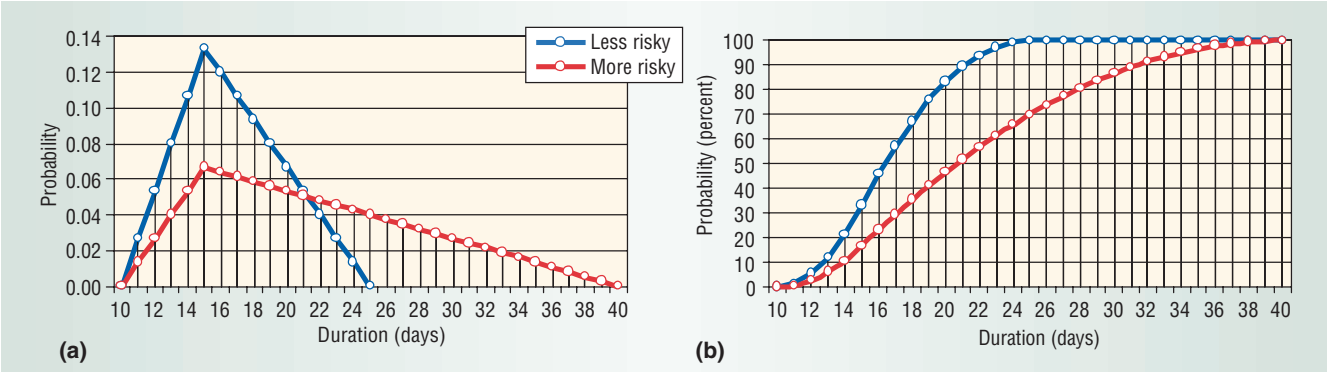


***Figure 3. On-time probabilities. Two tasks with the same earliest and most likely but different latest-completion dates have different levels of risk. (a) The expected completion date for the less risky task is 17 days, while the completion date for the more risky task is 23 days. (b) The on-time probability of the most likely date is around 37 percent in the first case and less than 20 percent in the second.***

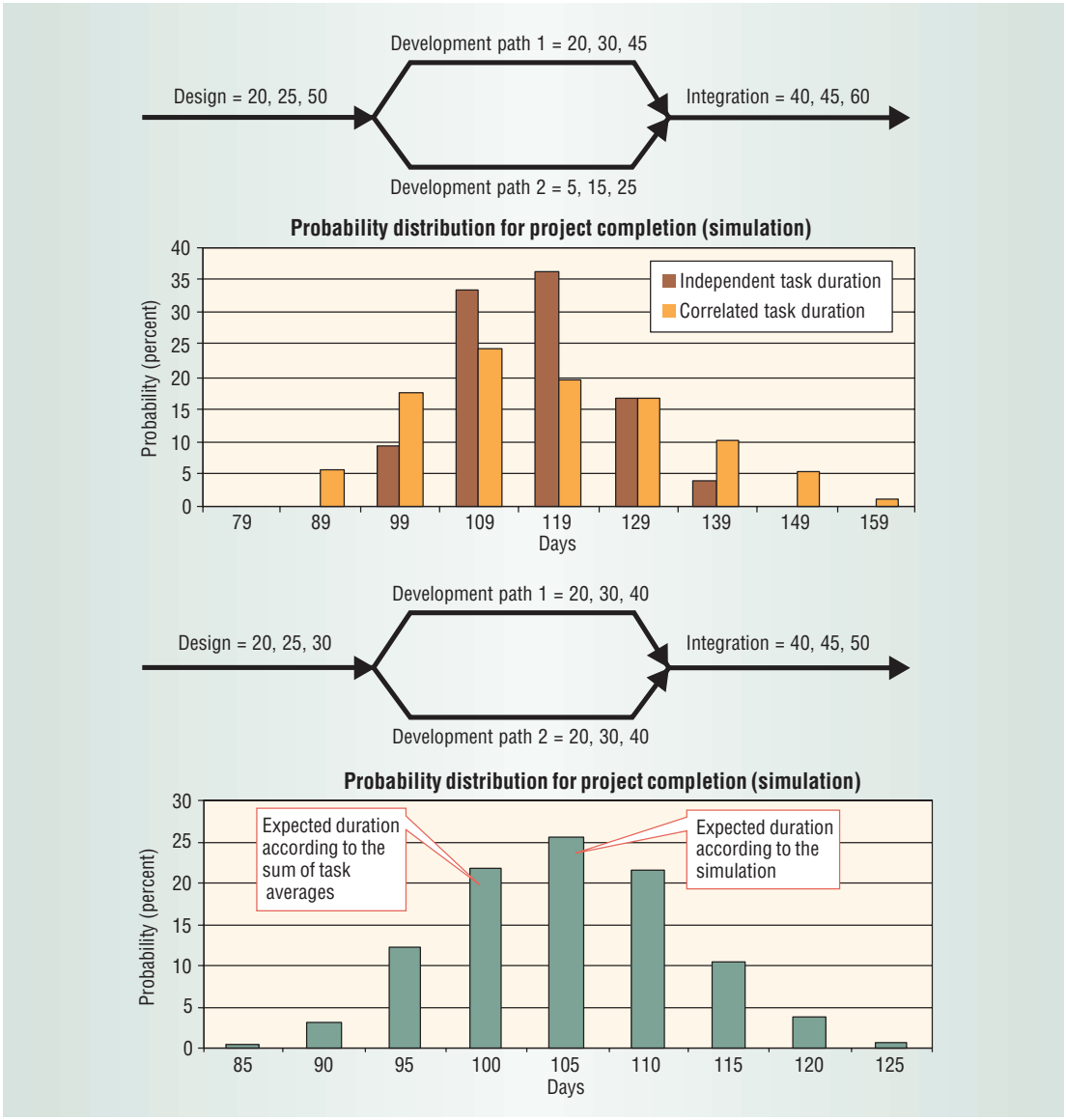


***Figure 4. Merging paths. In the presence of uncertainty, the expected project duration does not equal the sum of the expected task duration in the critical path.***

However, assuming independent task durations—a requirement of the central limit theorem—is perhaps one of the most dangerous assumptions a project manager can make. In practical terms, assuming independent task durations expresses the common belief that the late completion of some tasks is compensated by the early completion of others, so that, in the end, everything balances out. This assumption might be valid in certain industries (such as construction) or when dealing with certain events (such as rain), but it is particularly damaging in software development.

In a software development project, if one task is late because the complexity of the system was underestimated or the caliber of the team overestimated, it is highly likely that all tasks sharing the same underlying cause will be late, too. Thus if task durations are correlated through a common cause, the balancing process does not take place.

From a statistical perspective, dealing with correlated variables has two important consequences: First, the resulting distribution does not tend to have the bell-shaped curve that characterizes the normal distribution; second, the standard deviation tends to be larger than in the case of the sum of independent variables. From a practical perspective, this translates into an expected project duration that is located to the right, timewise, of that assumed under a normal distribution and into a higher uncertainty.

Another problem that traditional critical path calculations do not address is merging paths. As Figure 4 shows, when two or more tasks merge into an integration task, the earliest start of the integration task always corresponds to the late finish of the last completed precursor task. The merged path results in a mechanism that passes on delays but seldom passes on savings.

## MEASURING PROGRESS

When measuring a task's progress in terms of its main output—such as defined requirements, lines

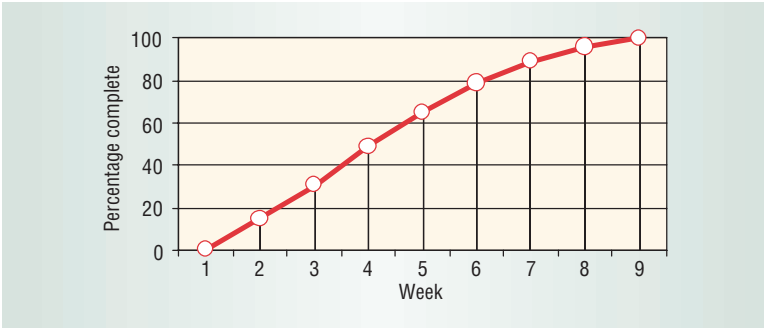


***Figure 5. Example of a production task for which the output does not grow at a constant rate. At the peak of productivity, between weeks three and five, the percent complete soars 20 percent in just one week. Toward the end of the task, it takes significantly more time to go from 80 percent to 100 percent complete.***

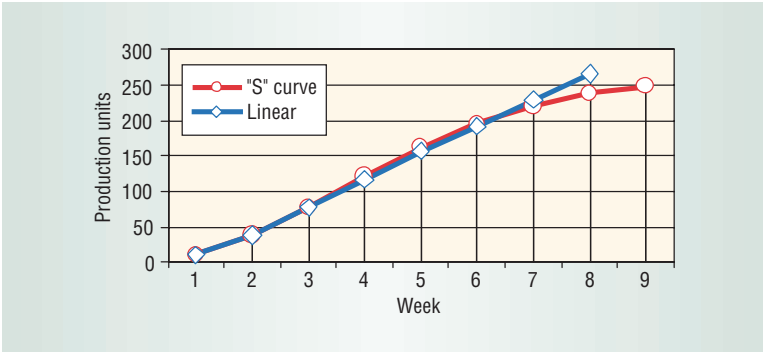


***Figure 6. Linear versus "S" forecasting. If the assumed task output is 250 production units, the linear projection forecasts the task's completion in week seven; the S curve, on the other hand, projects completion in week nine.***

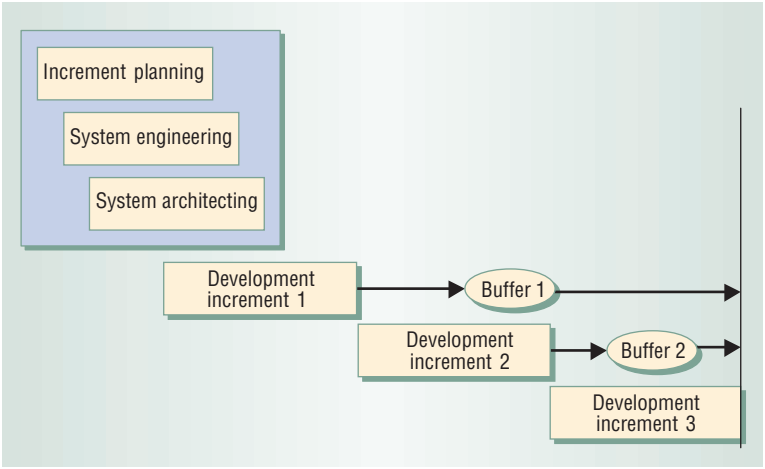


***Figure 7. The SPID project model. All increments (but the last) are isolated from the project delivery date by buffers designed to absorb most overruns.***

of code produced, discovered errors, or pages of written documentation—the output rate does not remain constant through the task's duration. The output rate more closely resembles the S-pattern[6-10] shown in Figure 5. This pattern, typical of many intellectual activities, is explained by the many actions and processes such as learning, team formation, work reviews, and corrections that occur at the beginning and end of a task, which consume time but do not contribute directly to the measured output.

Regardless of the factors behind this effect, it occurs so commonly that it even has a name: the 90-percent-complete syndrome. Ignoring this fact and extrapolating completion dates from the rates of progress observed through the half-life of a task using a straight line leads, almost invariably, to the announcement of optimistic completion dates that are never met. Figure 6 shows the error incurred by using a linear forecasting model instead of the S-curve paradigm.

## SPID APPROACH

Figure 7 shows the SPID project model, which entails increment planning, system engineering, system architecting, and two or more development increments.

In the increment-planning task, statistical techniques break down the project scope into a series of development increments in such a way that it is almost certain that all requirements allocated to the first increment will be implemented on time, that there is a fair chance to implement those allocated to the second increment, and so on. The system-engineering task employs requirements, value, and tradeoff analysis from a user perspective to set the project's priorities. The system architecting task creates the solution's general form, including the interface definitions and the analysis of dependencies between requirements.

All three activities take place concurrently to balance what the user needs and wants with what could be done from a technical perspective. Each *development increment* is a self-contained miniproject. SPID does not impose any particular approach beneath this level, which means that the team can organize development according to a waterfall model or an iterative cycle according to what is most appropriate for the project. A buffer that absorbs any overrun in their execution isolates all but the last increment from the project delivery date.

During execution, the team uses models to forecast work progress that more closely resembles the way people work than simply extrapolating from last week's progress reports. As Figure 8 shows, the team uses the output from the models to forecast completion dates and to take corrective action if necessary. The team doesn't start working on one increment until it completes the previous one. This prevents the team from wasting time developing things they might never finish anyway.

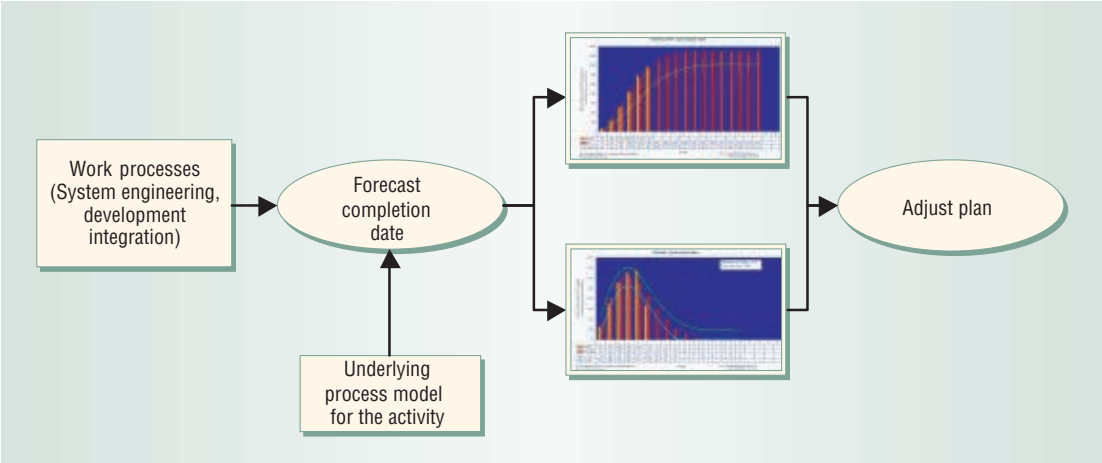


*Figure 8. The SPID control model. The development team uses the model output—rather than simply using the previous week's progress reports—to forecast completion dates and take corrective action if needed.*

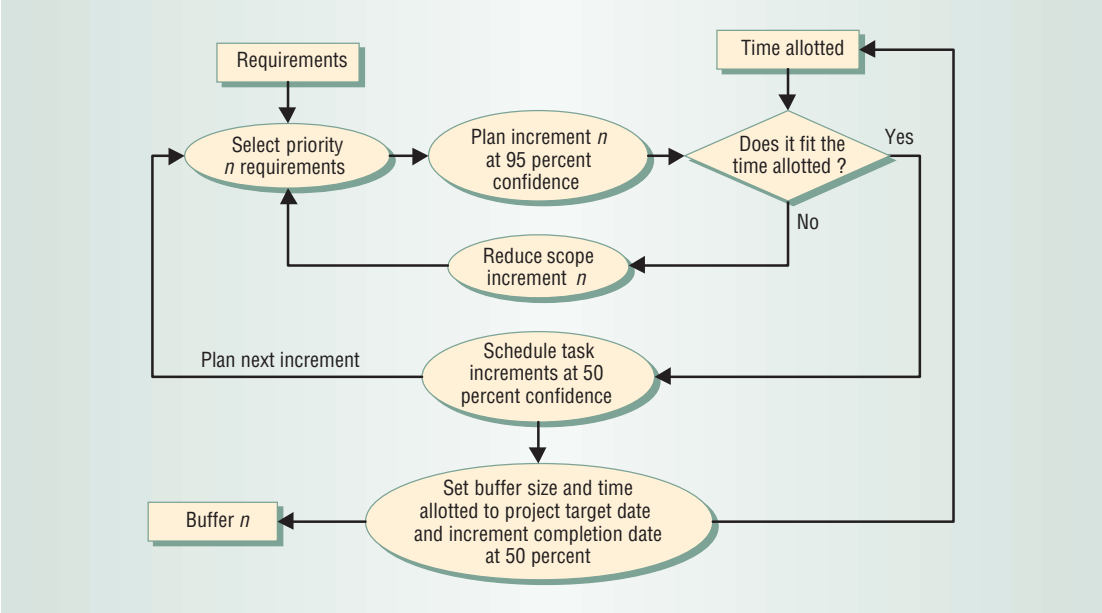


*Figure 9. The SPID planning process. The steps in the process are repeated for each increment the project includes.*

## Project planning

Once the development team establishes the feasibility of its project, it defines the duration of the development tasks in terms of their best, most-likely, and worst-case scenarios as a function of the increment's scope. The next step is to adjust the content of the first increment so it will have a high (roughly 95 percent) probability of being completed in the allotted time. Third, the team reschedules the tasks using the duration that corresponds to a 50-percent-on-time probability, allocating the difference between the high and low confidence dates to a buffer. Finally, the team plans the next increment using the length of the buffer as the time allotted. The process is repeated for as many increments as the project includes. Figure 9 illustrates the overall process. The probabilities and target dates could be calculated manually or by using one of the many commercially available project simulators.

In selecting requirements for each increment, the team must consider the need to provide functionally complete subsets to the user and the technical dependencies that exist between them. Table 1 shows the approximate probabilities for delivering the content of each increment when using SPID.

Although the specific techniques for estimating the duration of each increment development depend on whether the development team uses a cost model, an expert approach, or a Delphi process, including all reasonably expected completion dates between the minimum and maximum durations is crucial. If the minimum and maximum durations do not represent realistic alternatives, the whole exercise is futile. In the case of a parametric cost model like Cocomo, developers could do this by changing the value of key cost drivers. In the

**Table 1. SPID success probabilities.**

| Increment | Calculation | On-time probability |
|---|---|---|
| 1 | As planned | 95.0 percent |
| 1 + 2 | 0.50 * 0.95 | ~ 47.5 percent |
| 1 + 2 + 3 | 0.50 * 0.50 * 0.50 | ~ 12.5 percent |

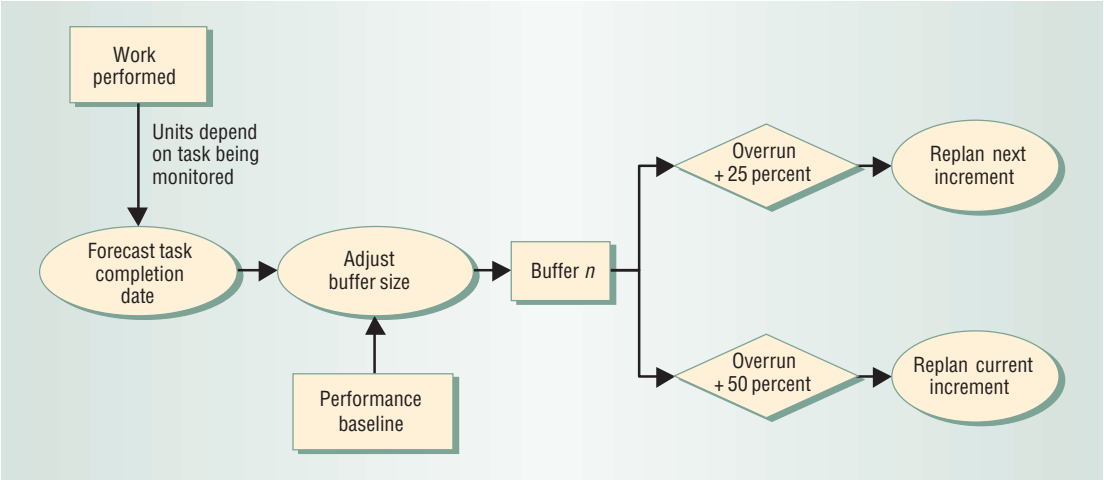


***Figure 10. The SPID control process. SPID exercises project control by updating the buffers with estimates at the completion of individual tasks.***

case of a Delphi process, this requires recording the optimistic and pessimistic estimates, not just the converging value.

## Project control

Time-bound projects provide little room for recovery, which means that once a problem manifests itself, it is almost too late. Controlling a project in these circumstances requires a mechanism that can identify the early signs of delay, minimize false alarms, and minimize disturbances to ongoing work.

The critical chain approach exercises project control by monitoring buffer consumption, but SPID updates the buffers with the estimates at completion of the individual tasks. To compute these estimates, the team could, for example, fit a Rayleigh curve to the progress reported and then project it into the future. Figure 10 illustrates the overall process.

Using buffers to monitor buffer consumption helps isolate the development team from overreacting to small variations and prevents false alarms that disturb the ongoing work. If replanning is needed, the team should take into consideration whether the underlying factors that affected the current increment's development will have an ongoing effect in the next increment. If so, the team should make appropriate cuts and not base its entire recovery strategy on the belief that it will do better next time.

## REWARDS AND RECOGNITION

How can all project stakeholders ensure that the development team will apply their best effort toward implementing all requirements rather than just getting by with implementing only the requirements in the first increment? The answer lies in the rewards and recognition system. Whether employee rewards or price incentives in contracts, the incremental model provides clear criteria for evaluating performance.

No reward is associated with delivering the first increment. All team members simply do their jobs. Subsequent increments result in increased recognition of the extra effort put into the task. The on-time probabilities shown in Table 1 provide a means for calculating the reward's expected value. This calculation is important because a large amount—with a very small probability—will result in a low expected value. In this case, the team members might perceive this reward as a lottery they are unlikely to win; consequently, the reward loses much of its motivation value.

As an example, a $5,000 reward for "Increment 2" has an expected value of $2,375. Applying the same amount to "Increment 3" has an expected value of $625. Clearly, the motivational value of the reward is not the same in both cases. The same ideas could be used in pricing a contract or in a bidding process. Delivering the requirements included in the first increment will result in the contractor receiving a nominal fee, while delivering the requirements included in the other increments will entitle the contractor to premium payments.

In addition to bringing together several existing project management techniques, SPID also provides a decision rule to calculate increment size, a reward model based on the expected value of the increments, and a recommendation to track the project based on forecasts rather than on actual progress.

SPID's basic premise is that businesses are better off when they know what they can expect than when developers promise them the moon but offer no credible assurances about when they can deliver the project. SPID recognizes that in any development project, hundreds of things can go right and thousands of things can go wrong; SPID makes these factors an intrinsic part of the planning and control processes. ■

**Acknowledgments**

I thank Tamara Keating, Ericsson Research Canada; Alain Abran, University of Quebec at

Montreal; and Raul Martinez, RMyA, for their assistance in preparing this article. I also thank the anonymous reviewers for their helpful comments and insight.

## References


1. E. Goldratt, *Critical Chain*, North River Press, Great Barrington, Mass., 1997.
2. R. Newbold, *Project Management in the Fast Lane*, St. Lucie Press, Hampton, N.H., 1998.
3. S. McConnell, *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, Redmond, Wash., 1996.
4. N. Pisano, "Technical Performance Measurement, Earned Value, and Risk Management: An Integrated Diagnostic Tool for Program Management," http://www.acq.osd.mil/pm/paperpres/nickp/nickpaso.htm (current Feb. 2002).
5. S. Grey, *Practical Risk Assessment for Project Management*, John Wiley & Sons, New York, 1995.
6. K. Pillai and S. Nair, "A Model for Software Development Effort and Cost Estimation," *IEEE Trans. Software Eng.*, vol. 23, no. 8, 1997, pp. 485-497.
7. L.H. Putnam and W. Myers, *Measures for Excellence: Reliable Software on Time, within Budget*, Prentice-Hall, Upper Saddle River, N.J., 1992.
8. J. Martino, *Technological Forecasting for Decision Making*, McGraw-Hill, New York, 1993.
9. E. Miranda, "The Use of Reliability Growth Models in Project Management," *Proc. 9th Int'l Symp. Software Reliability*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 291-298.
10. J. Gaffney, "On Predicting Software Related Performance of Large-Scale Systems," *CMG Proceedings*, Philadelphia, 1984.


***Eduardo Miranda** is a senior specialist at Ericsson Research Canada and an industrial researcher affiliated with the Research Laboratory in Software Engineering Management at the University of Quebec at Montreal. His research interests include project management, estimation techniques, and software measurement. He received an MEng in engineering management from the University of Ottawa, Canada, and an MSc in project management from the University of Linköping, Sweden. Miranda is a member of the IEEE Computer Society and the ACM. Contact him at eduardo.miranda@ericsson.ca.*