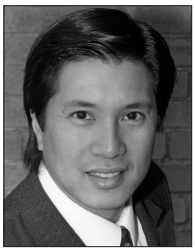
IT METRICS STRATEGIES
Helping Management Measure Software and Processes and their Business Value



Internet Speed Deadline Management: Negotiating the Three-Headed Dragon

by Michael Mah

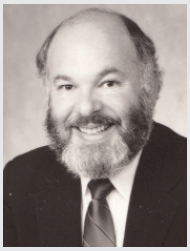
The deadline rules. This month, we take a momentary shift from IT benchmarking and tackle the management of projects under Internet speed deadlines. How can we employ metrics to deal with the pressures of ever-growing project scope under ever-tighter deadlines? What does this process do to IT productivity? Can we do better?

One thing we know: time pressure can either make a project or kill it entirely. Not understanding the dynamics of time pressure on the productivity of IT projects is something we simply can't afford.

Managing the deadline requires adroit skill on the part of several parties with both common and clashing interests. Multiple parties, multiple interests, a competitive marketplace. Negotiation needs to come into play. Specifically, it's vital to satisfy several interests and the inherent conflicts in an integrative, win-win frame versus a distributive, win-lose frame.

Skillfully navigating this terrain will provide an IT organization with incredible opportunities to capitalize on success in ways another company may not. It's all in the skill of balancing the needs and interests of several entities in an organization, from marketing to the IT staff to the CIO. Simply imposing a dictated deadline to a hapless IT group with its back against the wall will not work.

Continued on page 2.



When the Project Absolutely Must Get Done: Marrying the Organization Chart with the Precedence Diagram

by Stan Rifkin

What should you do when a project absolutely must be done — when you have no alternative but to accomplish it, no matter how uneasy you feel about its prospects? What you need is a tool that goes beyond typical macro-estimation to model this single, absolutely-must-get-done project.

I recently had the chance to use a new tool, ViteProject, which takes the estimates from a macro-estimating process and seeks to model an individual project. It helped us (me and my client) concisely match the organization of our resources (namely, staff) with the organization of the project. It helped us determine the optimal

Continued on page 8.

executive summary

A driving factor in what makes managing IT projects so hard is the fact that so many teams are given a deadline *first,* with very little definition of what it is they have to do when it comes to defined requirements. How do you deal with that? In my article, "Internet Speed Deadline Management," I take on this subject, exploring how to focus the organization into solving that fundamental common problem.

Stan Rifkin of Master Systems then takes us into research that has emerged from 10 years of work at Stanford University. His article, "When The Project Absolutely Must Get Done," deals with alignment; specifically, the alignment of critical skills of the organization to various phases of the project — exactly when it needs them.

Taken collectively, I hope these two perspectives can be brought to bear and enable the application of metrics to a critical issue — to help new projects succeed. That is, after all, what companies pay IT people to do.

Michael C. Mah, Editor

may 2000 vol. VI, no. 5

CUTTER CONSORTIUM

schedule is about 2.5 times harder than cutting effort and cost. The same is true for quality. When project size and scope grows, all three behave in what seems like unpredictable ways. But it is predictable for those who understand this three-headed dragon. Once you understand these fundamentals, you can begin to address the interests of IT developers, management, end users, and the company itself, and start solving the common problem of living in this Internet speed e-commerce world.

Apply metrics to solve a problem of the deadline by negotiating functionality *backward.* This leads to a dialog on agreeing to project scope that is adequate enough to seize an opportunity in the market, yet manageable enough to be built within a tight market window. This will enable a team to have the time it needs to build an application with adequate time for testing, ensuring that a system will operate reliably when placed into service. If that happens, everyone wins.

References:

[1]Based on informal surveys of IT developers and managers, most recently at *SM/ASM 2000*, San Jose, California, USA March 2000.

[2]Based on informal trade conference surveys, in which virtually none of the attendees report the existence of sustained metrics programs that are used for strategic IT decision-making.

[3]Lewicki, Roy J., Joseph A. Litterer, John W. Minton, and David M. Saunders, "Strategy and Tactics of Integrative Negotiation." R.D. Irwin, 1985.

[4]Kelman, Herbert, "Negotiation as Interactive Problem Solving," *International Negotiation*, Vol. 1., 1996.

When the Project Absolutely Must Get Done

Continued from page 1.

combination of resources and work so that we did not have to guess, and it predicted the impact of our choices.

ViteProject is the result of 10 years of research at Stanford University (Stanford, California, USA) into how to best optimize a particular project. It begins with a macro-estimate, a chart showing the precedence of the major activities and milestones, and an organization chart of the project. With this and a little more information, we were able to examine tradeoffs and their impact on duration, cost, features, and quality. For example:

- What is the effect of adding a deputy software project leader? Does the additional expense justify the reduction in duration?
- What is the effect of placing subject matter experts or more experienced software engineers on the project? Is it more cost-effective to have expert project management or expert programming?
- If the prerelease error rate doubles, what is the right tactic: drop everything and fix errors? Or keep developing, fixing, and testing?
- What amount of e-mail and voice mail messages are missed at the height of the project?
- What is the proportion of work, rework, and idle waiting time in each scenario?
- Are any particular process improvements more cost-effective than another on this project?

Estimating Duration

What makes knowledge teamwork different than, say, building construction teamwork? What makes it difficult to estimate the former and perhaps not the latter? One view is that knowledge teams face more uncertainty because of the intangible nature of their work — less is physical and visible, therefore less is known and more is uncertain. This description of knowledge work is basically what Jay Galbraith calls the information processing problem: "The greater the uncertainty of the task, the greater the amount of

information that has to be processed between decision makers."[1]

Traditional design of the organization structure is an artistic balance that strives to match the information processing requirements, which are predominantly contingencies, to the information processing capacity of the structure. Accordingly, in the traditional practice, the externalities are given and our job is to artfully design a structure that matches or fits them. We measure the success of our organizational design job based on how we define "fit."

The creation of a new field of study, computational and mathematical organization theory, has called into question such art. It tries to provide more information so that (in our case) organization structure can be more objectively measured and optimized. For example, what if you could optimize the design of the team, and thereby the structure of an organization, such that you can discover the design that gets more work accomplished? Wouldn't this provide a more tangible measure of whether we had achieved our objective?

A Different Way to Estimate

Staff members of the Center for Integrated Facilities Engineering in the Civil Engineering Department in Stanford University's School of Engineering read everything written on project management to try to understand and estimate the work of engineering design teams. The Stanford researchers found nothing in the project management literature that they could use, nothing that explained the variation in team performance they had observed. They turned to the literature on organizations and found that contingency theory and its information processing view resonated with them. Their challenge was to reify the constructs in such a way that values could be verified during field tests on real projects.

The idea struck them that perhaps they could model an organization at work by simulating the passage of work "packets" through the network of activities described by a traditional task precedence diagram and linking each activity to the actor responsible for that activity. This way, elements important to project success could be made visible. In other words, this would translate contingency theory constructs into concrete project variables. And the notion of "fit" would be reified to normal project success criteria, such as duration, cost, and quality.

The Stanford researchers first developed Virtual Design Team, to be used in educational settings, and then ViteProject, for commercial use. The displays presented here are from ViteProject.

How ViteProject Creates Simulations

Here are some examples of how ViteProject models various workflow scenarios.

Modeling "change propagation" or "failure dependence." When projects are "fast-tracked" — that is, composed of several parallel, simultaneous activities — information (or defects) found in downstream activities needs to be communicated to other (possibly still upstream) dependent tasks. In other words, the simulator needs to make work that is not on the original precedence diagram, work that we all know from experience will arise. ViteProject does this by creating activities composed of subactivities. The simulator assumes there are 20 subactivities per activity and that each subactivity takes 1/20 of the total effort. At the end of each subactivity, the simulator checks to see if it's possible that an exception occurred. This is based on a percentage (hopefully based on field measurement) entered by the user/modeler. If an exception has occurred, the simulator decides at random whether the failure should be reworked, quickly "patched," or ignored. Only the first alternative substantively changes the product, but at the expense of effort. If there are many selections of the last two alternatives (patched or ignored) then product/outcome quality would be questionable.

Modeling centralization. Centralization refers to the degree to which exceptions are handled centrally versus locally. If exceptions are handled centrally, often there is work not shown on the precedence diagram to communicate and propagate the exception conditions upward and the decisions downward. Again, the simulator will be asked to insert work where none was hitherto indicated. If a project has a high degree of centralization, it is possible that the managers up

the hierarchy get backlogged and cannot respond in time, a delegation by default. In other words, programmers at the bottom of the hierarchy only wait so long for a centralized decision up the hierarchy and it it takes too long, they proceed based on their own local best guess. Central decisionmakers tend to have greater expertise and a broader context for making decisions, so the quality of their decisions is likely to be better than local ones. So, the simulator now has numerous behind-the-scenes tasks: generate work that is not shown on the precedence diagram, model in-baskets so that backlog can be illustrated, and modify decision quality based on the place of the decisionmaker in the hierarchy.

Modeling formalization. This contingency variable refers to "the likelihood that an actor who needs to exchange information with another actor will wait for a formally scheduled meeting versus initiating an ad hoc communication seeking the information needed."[2] Again, the simulator has to create work not on the precedence diagram to model the level of communication requested. This is done primarily by affecting the in-baskets of those with whom communication is desired.

Creating a Base Scenario

The first step in using ViteProject concepts is to establish a base scenario, from which other scenarios will be simulated. The organization used in this example is a financial services institution that uses a standardized project development method. I have simplified the project slightly for the sake of clarity. We begin with Figure 2, which shows the hierarchy of exception handling on the project. Figure 3 shows a precedence diagram of activities. This is an excerpt of the standard software development project method. The milestones run along the top, and the activities to achieve each milestone are performed in parallel between each milestone. This example assumes that 50,000 lines of code need to be developed to deliver specific functionality for a business application. We have made additional assumptions about the process maturity of the team and enveloping organization, and we have made an assumption about the maximum rate at which staff members are available for the project and the rate at which they can be absorbed. Our goal is to optimize the fit between the organization structure and the work to be accomplished.

The first step in establishing a base scenario was to use macro estimation to determine the overall duration, the duration and effort for each milestone, and the peak number of staff required for each milestone. We then allocated the effort between milestones to individual job types in accordance with the real project this one mirrored. The next step was to connect a responsible party to each activity. In Figure 4 (on page 12), the barely readable figures on each arc from an actor

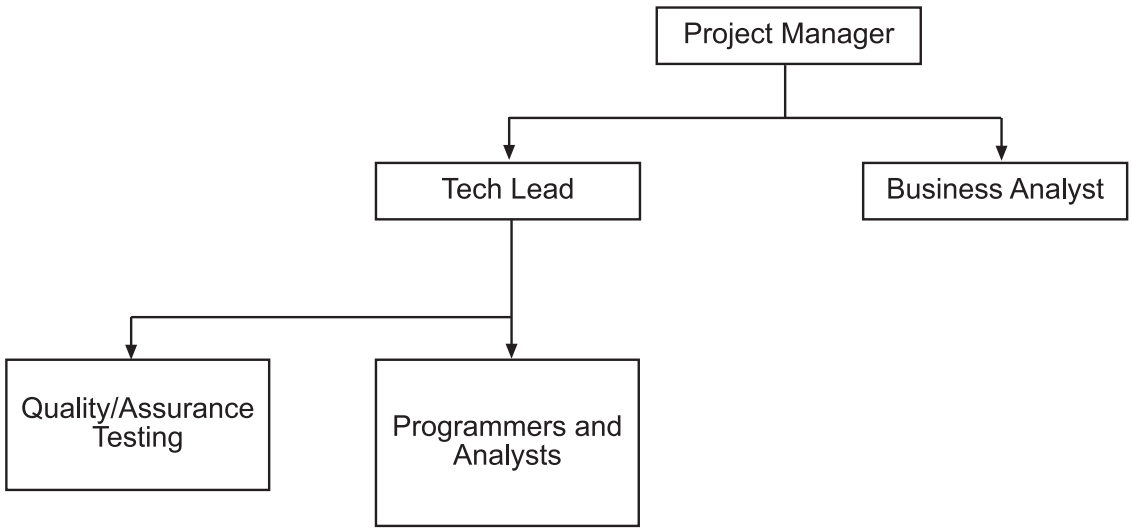


Figure 2 — Exception-handling hierarchy.

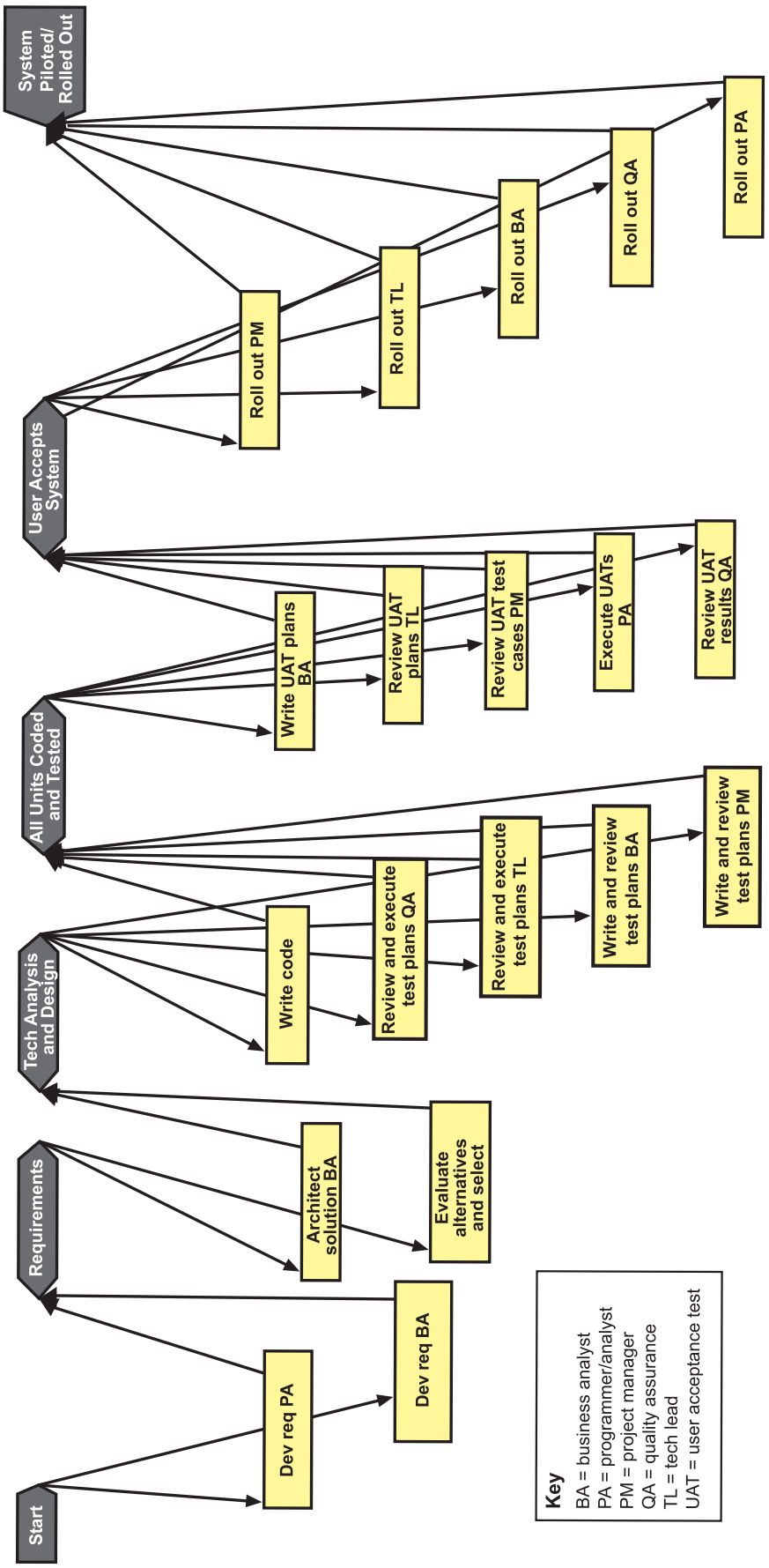


Figure 3 — Precedence of activities.

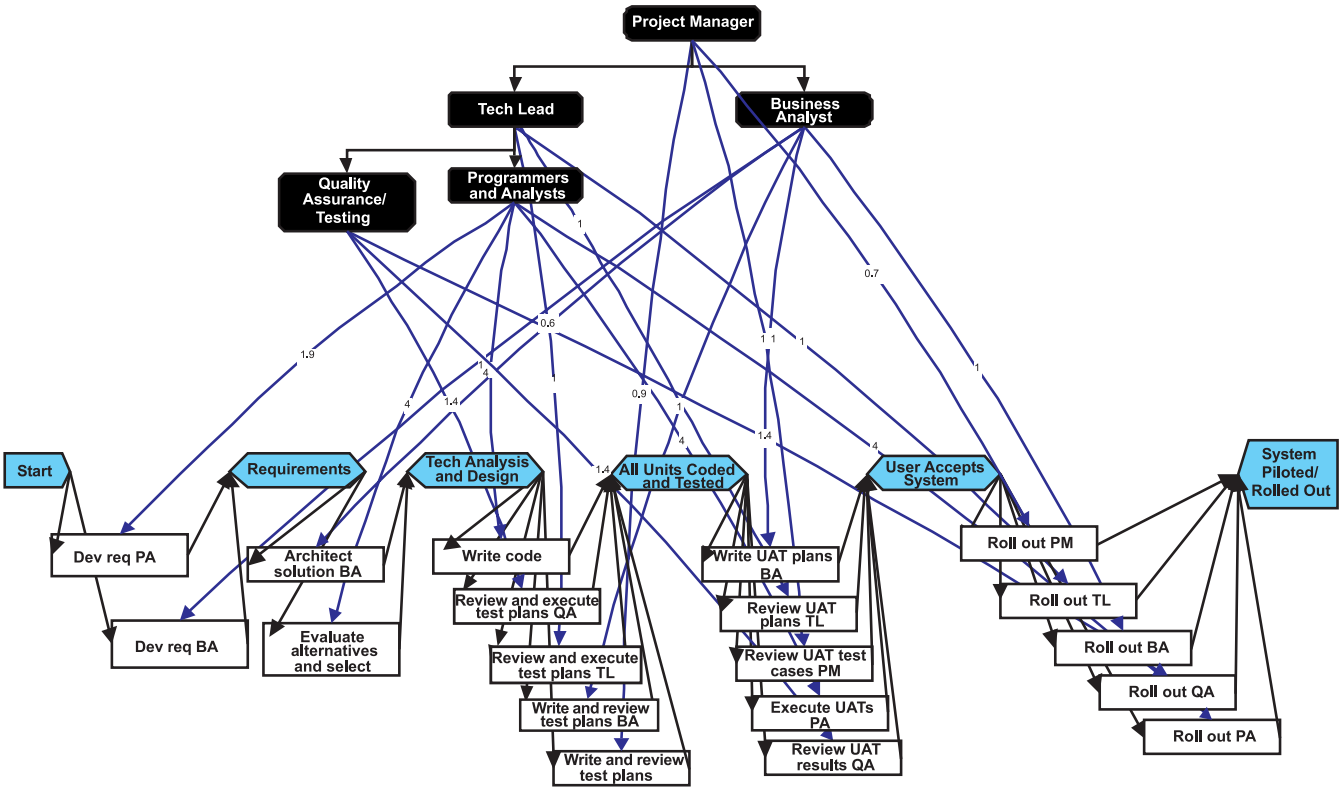


Figure 4 — Connecting exception handling with activities.

to an activity are the number of full-time equivalent actors assigned to an activity. In accordance with the actual project, we had a pool of actors, of which only a subset were assigned to a particular activity at any one time.

Next, we used ViteProject to simulate the workflow implied by this configuration of actors and activities as a way to establish the ViteProject base case. This gave us Scenario 1, in which the project would take 380 days and cost US $605,400 (See Table 1). We saw that the difference between ViteProject and the macro-estimate was negligible, so we knew that we had set up the ViteProject base scenario properly.

Scenario Tuning

Macro-estimation tools "generically" build in some of the variables that ViteProject considers explicitly, such as error rates between activities, communications overhead, the effects of experience in roles, and the effects of centralization and formalization.

Centralization is the degree to which exception handling is conducted up the hierarchy, as opposed to locally, nearest to the place in the organization where the exception arose. In software projects, exception handling is typically responded to locally, not centrally. As this was the case in our sample project, we adjusted ViteProject's setting from medium (the default) to low. Similarly, in software projects, communication formalization is typically low; therefore, we adjusted ViteProject's setting to low.

We also adjusted the experience levels from ViteProject's defaults. For this particular project, the project leader had high experience and the quality assurance team had low experience. Making these adjustments gave us Scenario 2, in which the project would take 450 days and cost $553,800 (see Table 1).

Scenario 3 in Table 1 shows what happened when we adjusted the functional error probability rate to 20%. The 20% figure reflected actual project experience in this case. It means that one out of every five subtasks

internal to an activity had to be reworked. The identification of such errors might be from inspections or testing.

Calibration

The combined effects of adjusting for centralization, formalization, experience, and functional errors increased the duration of the project by one-third. This is because macro estimation represents the effects of centralization, formalization, staff experience, and error rate as a single item: work. Our task is to tear apart the work unit into its four constituents — real product work, rework in light of errors, coordination time, and time to wait for decisions — so that we have the flexibility to model each one separately. The increased duration is the effect of explicitly modeling what the macro estimate calls "work" into its four parts, so we reduce the effort by one-third to get back to the baseline estimate. That is, we have set the ViteProject "work" to be two-thirds of the macro-estimate and the sum of rework, coordination, and waiting time is the remaining one-third.

Scenario 5 was our final attempt to get close to the original macro-estimate by adding the effect of project errors at the rate of 20% for those tasks likely to require extensive rework. The 20% figure came from actual measurements at the organization under study. This brought the simulated duration to 383 days (versus 386 days from the macro-estimate). The three-day difference was insignificant for our purposes.

What-If Questions

Before we began trying to optimize the project, we examined a question the project leader had asked us: what would be the effect if the prerelease error rate were double the typical figure and what would be a reasonable antidote? The project leader asked this question because he had early indications that the prerelease error might, in fact, become 2X the normal experience.

Scenario 6 shows the effects of doubling the functional error rate. Essentially, the duration increases by about 17% and the costs by about 19%. The project would slip about 20% of its schedule and effort.

In searching for an antidote, we looked at which type of actor consumed the greatest duration. As Figure 5 indicates, programmer/analysts are by far the most used actors. What would be the effect of making them more efficient by increasing their experience to a higher range within the same pay scale? This would be equivalent to selecting programmers in the pay range with the greatest process maturity (assuming they were available).

In Scenario 7, we increased programmer/analyst experience from the default of medium to high. The result was to get back virtually all of the loss of a 2X prerelease failure rate. (The increases in work rates from medium experience to high were estimated from actual project experience.)

Table 1 — Scenarios Created in ViteProject

Scenario	CPM Duration	Simulated Duration	CPM Cost	Simulated Cost	Description
1	386 days	380 days	605.4K	605.4K	Baseline
2	455 days	450 days	553.8K	553.8K	Centralization and formalization low; experience high and low
3	455 days	510 days	553.8K	640.1K	20% functional (internal) error probability
4	338 days	381 days	384.8K	454.4K	$^1/_3$ reduction in FTEs required
5	338 days	384 days	384.8K	483.3K	Added failure dependencies and 20% rate
6	338 days	445 days	384.8K	568.6K	Double prerelease failure rate
7	290 days	384 days	323.4K	479.6K	Increased skill of programmers for 2X prerelease failure rate
8	338 days	384 days	384.8K	485.4K	Normal failure rate; removed milestone
9	298 days	331 days	384.8K	484.3K	+1 quality assurance analyst and 1 programmer/analyst
10	280 days	313 days	384.8K	486.3K	=1 programmer/analyst during most intense coding
11	278 days	306 days	378.3K	480.9K	Increased tech lead skill and pay

Optimization

The last four scenarios show the most interesting phase of our work with ViteProject: looking at how to reduce duration and costs by improving efficiency and quality.

Our first thought was to remove the integration test milestone because it synchronizes according to the activity with the longest duration. Optimizing the interdependencies between code integration and system testing should have helped trim the time. For example, those writing test cases do not have to wait for a synchronizing activity like completion of a milestone to proceed to work on the next milestone.

Because we still had to wait for the slowest performers, removing the integration test milestone actually had no effect (see Scenario 8). The overlap we expected to achieve did not materialize because the actors with the longest duration during integration testing are also those with the longest duration of the next activity: system testing.

We next looked at communication backlogs, knowing that communication profoundly affects quality, and that quality reverberates throughout a project. In Figure 6, we see that programmer/analysts and quality assurance (QA) analysts are missing three to four days of messages during the middle of the project (centered around July). We added a programmer/analyst and full-time QA person during those peak periods to see the effect. The results (Scenario 9) are dramatic: the duration decreases by about 50 days.

(Note that adding such resources reflects two fundamental assumptions that are realistic for this modeled project. First, resources are available for relatively short durations. Second, there is no significant ramp-up or learning-curve time. This is realistic in this case because the work was subdivided in such a way that application-specific knowledge was not required for some tasks in the short run.)

However, programmer/analysts were still backlogged around the second week in May. We looked at the Gantt chart and saw that this was during coding, so we added another programmer to that period alone. The result is Scenario 10, in which we gained another 18 days of schedule for about $2,000 increase in cost.

Looking at Figure 7, we see that the technical lead is now backlogged. We increased the skill level of this position to see the effect, shown in Scenario 11. This scenario saved another seven days of duration. It also saved about $5,000 because the tech lead's

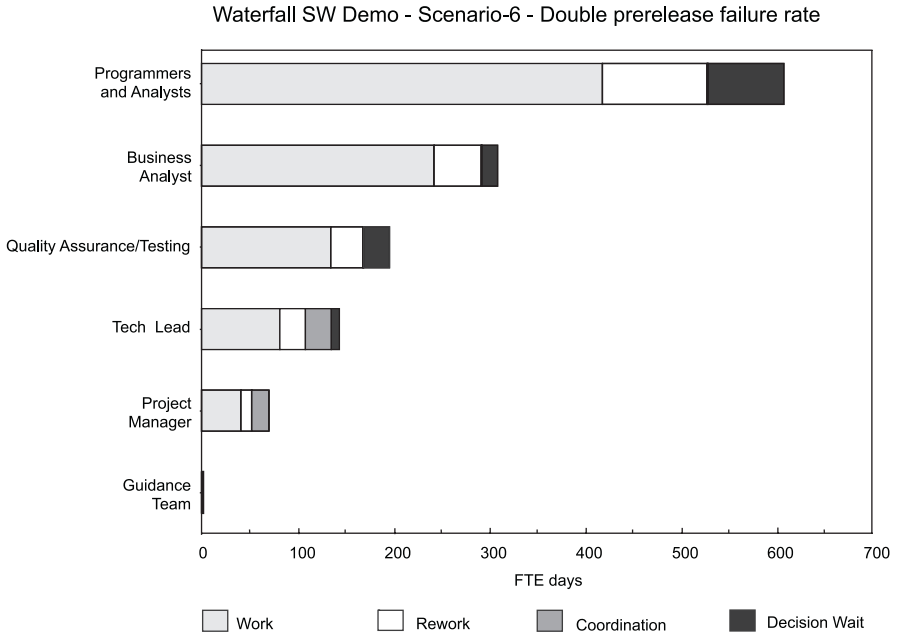


Figure 5 — Actor work breakdown.

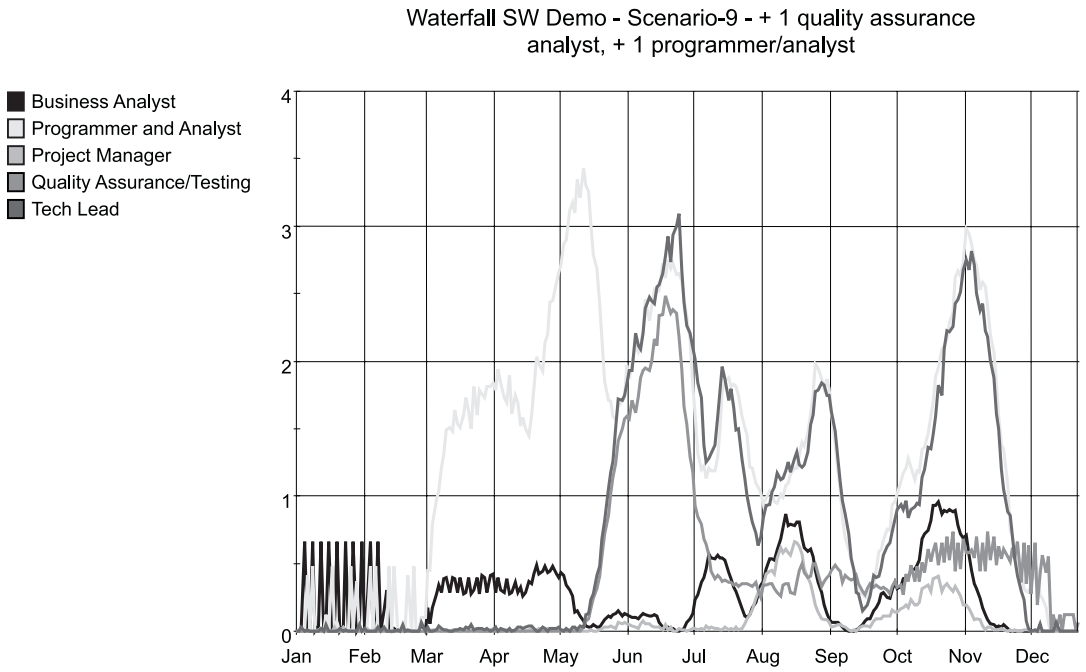


Figure 6 — Actor backlog.

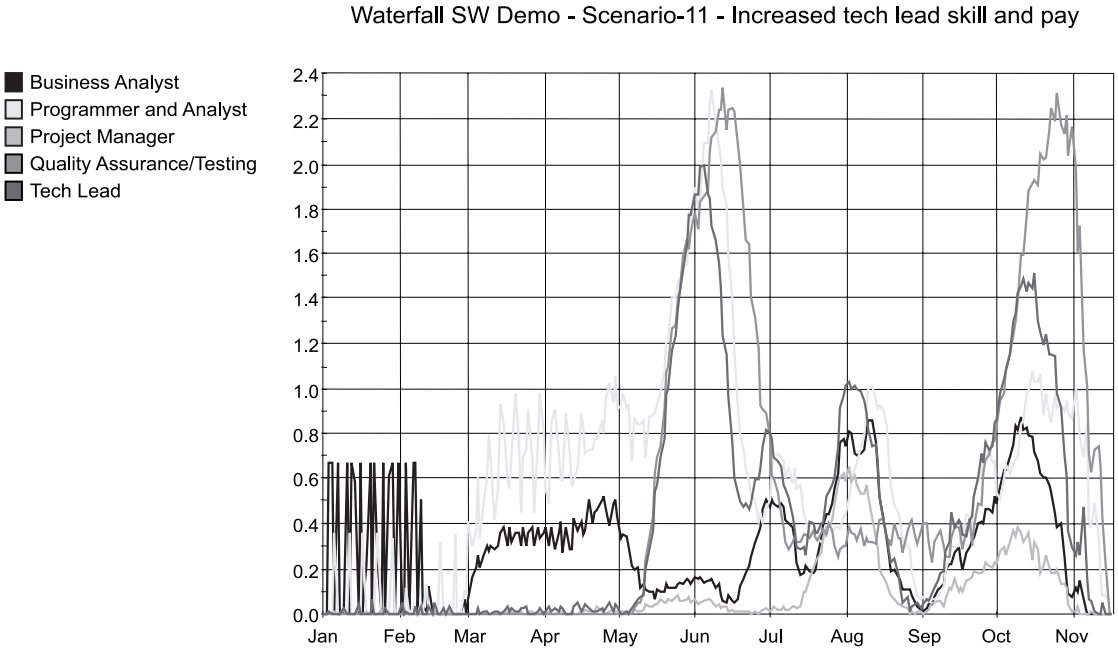


Figure 7 — Revised actor backlog.

subordinates don't have to wait so long for decisions.

Conclusion

The macro-estimation model forecasted that the project would take 386 days at a cost of $605,400. Using VitеProject, we were able to reduce this to 306 days and $480,900. This represents a savings of about 20% in both schedule *and* cost by optimizing our staff alignment using computer modeling, thereby "tuning" our productivity to higher levels.

VitеProject demonstrated that we can achieve the lower bounds of the duration envelope predicted by macro-estimation tools by paying attention to the interaction between how we organize our team and how the work is laid out.

Acknowledgements

I gratefully acknowledge the pioneering work of Ray Levitt and John Kunz at Stanford, and of Bob Drazovich of Vité.

References

Galbraith, Jay. "Organization Design: An Information Processing View," *TIMS Interfaces*, No. 4, 1974.

Levitt, Raymond. *The VitеProject Handbook: A User's Guide to Modeling and Analyzing Project Work Processes and Organizations*. Vité, 1998.

Burton, Richard M. and Borge Obel. *Strategic Organizational Diagnosis and Design: Developing Theory for Application* (2nd edition). Kluwer Academic Publishers, 1998.

Jin, Yan, and Raymond Levitt. "The Virtual Design Team: A Computational Model of Project Organizations." *Computational & Mathematical Organization Theory*, Fall 1996.

Putnam, Lawrence H., and Ware Myers. *Measures for Excellence: Reliable Software on Time, Within Budget*. Prentice-Hall, 1991.

http://www.stanford.edu/group/CIFE/VDT/Publications.html.

About the Author

Stan Rifkin is a principal with Master Systems, Inc., an advisory services firm that concentrates on improving the processes by which organizations manage and develop software. Each year, *Wall St. and Technology Magazine* selects two CIOs of the year; last year Rifkin was the only consultant to have served both of these CIOs. He has worked at the Software Engineering Institute, where he was a project leader in the process improvement program. Rifkin's speciality is implementation and deployment of best practices. He can be reached at sr@Master-Systems.com.