When the project absolutely must get done: Marrying the organization chart with the precedence diagram

Stan Rifkin
Master Systems Inc.
PO Box 8208
McLean, Virginia 22106 USA
+1 703 883 2121
sr@Master-Systems.com

1 ABSTRACT

Very little is new in project planning, but this is! We present a technique to marry the organization chart with a project's task precedence diagram. This permits us to simulate the project at a micro, project-specific level never before achieved. We can perform "what-if" scenarios related to organization structures, the deployment of specific individuals and skills, and the structure of information flow and exception-handling in a project. The tool used, ViteProject, was developed over the last ten years in a Stanford University laboratory, where substantial results have been achieved when applied to design activities other than software. We present our real-world experience with several software projects, where it has improved project visibility and allowed us to rationally optimize projects in a way hitherto impossible.

Keywords

Project management, micro-estimation, project simulation

2 INTRODUCTION

When a project absolutely must be done, we have at our disposal few tools to help us optimize the project and experiment with options for success. Macro estimation tools, such as Quantitative Software Management's (QSM's) SLIM (Software LIfecycle Management) suite, helps us find the bands or range of practicality of our plans, and to its credit does not give us a point estimate but rather a range of, say, durations and associated probabilities of achievement. Our task is to organize the project so that it takes the lowest duration in the range, one that macro estimation tools do not address.

ViteProject is the result of ten years of Stanford University research into how to best optimize a particular project. It begins with a macro estimate, like that supplied by QSM's SLIM Estimate or COCOMO II, a chart showing the precedence of the major activities and milestones, and an organization chart of the project (it is not the authority hierarchy, but rather the exception handling hierarchy).

With this information and a little more, we can examine trade-offs and their impact on duration, cost, features, and quality. An example set of trade-offs, some of which will be explored in this paper, is:

1. What is the effect of adding a deputy software project leader? Does the additional expense justify the reduction in duration?
2. 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?
3. If the pre-release error rate doubles, then what is the right tactic: drop everything and fix errors or keep on developing, fixing, testing?
4. How much e-mail and voice messages are missed at the height of the project?
5. What is the proportion of work, re-work, and idle waiting time in each scenario?
6. Are any particular process improvements more cost-effective than another on this project?

3 ESTIMATING KNOWLEDGE WORK DURATION

Predicting the duration of a knowledge team's project work has always been a challenge. While there is a tradition of careful and accurate estimation for, for example, building construction duration, there is nothing of the kind for building design, essentially a knowledge team's work. One cannot use Gersick's admonition to set deadlines and then the team can work to them. [5] We have to have a way to know what reasonable deadlines are.

One can use the insight that there is an inextricable interdependence between structure of the work and structure of the team. That is, the structure of the work is optimal *relative* to the particular structure of the team, and the structure of the team is optimal *relative* to the particular structure of the work. Accordingly, one size does not fit

all, structures ought to be tailored to the work. We are taking an additional liberty, an additional degree of freedom, by stating that, symmetrically, the work should be structured according to the team. This last is the new bit.

What is the best way to organize? "It depends," answers the (contingency) theorist. "Depends on what?" The standard responses [1] are organizational complexity (horizontal, vertical, and spatial differentiation), formalization, centralization, degree of coordination and control, management and leadership style, organizational climate, size and skill capabilities, environment, technology, and strategy.

What makes knowledge team work different than, say, construction team work? 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, more is uncertain. This description of knowledge work is basically what 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." [4, p. 28].

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

The creation of a new field of study, computational and mathematical organization theory (CMOT) [2,3], has called into question such art. It tries to provide more information so that, in our case, organizational structure can be more objectively measured and optimized. For example, wouldn't it be attractive if we could design the structure of an organization such that given the externalities one design got more work accomplished than the other designs we could think of? Wouldn't this provide something of a more tangible measure of whether we had achieved our objective?

4 AIM

Accordingly, the aim of the research and application described here is to measure the structure of organizational design with respect to the specific work to be accomplished by the software organization under study. The work is described in the traditional way: a task precedence diagram of the activities and milestones to be accomplished and an estimate the effort required for each activity. And the organization is described in a traditional hierarchy chart, but this one describes exception-handling and not authority per se. One of the major breakthroughs (of the approach to be described) is to connect each person (actor) in the exception-handling hierarchy to the activities for which he/she is responsible. Once the topology of the project is mapped to the structure of the exception-handling, then a simulation of the work of the project team can be undertaken and the usual measures of project performance collected and compared: duration, verification risk (the risk that quality problems escaped into the work products, which is a surrogate for product quality), throughput, and cost. Now we can compare various configurations of the work with various configurations of the hierarchy and use data to guide our definition of "fit" or optimality.

The challenge is the simulation. One must carefully read the contingency theory literature and try to operationalize it. For example, the literature might suggest that formalization of routine tasks reduces the demand on the information processing mechanism of a structure, all other things equal. It makes sense on the surface that increased formalization is a method of spreading information down the hierarchy so that the hierarchy does not have to process information about tasks for which formal methods exist. But how much formalization begets how much decrease in information processing demands? And how are formalization and information processing demands measured and represented?

Computational and mathematical organization theory is an exciting field precisely because it addresses in concrete terms what we all think we understand based on discourse alone. While we may think we understand contingency theory, we lack variables sufficiently concrete to actually demonstrate our understanding and test hypotheses in real or simulated work situations. All of the contingency constructs mentioned so far are too ethereal to be able to posit and measure the values of variables in the field (even size, in the contingency theory sense, is about bigness and span of control, and we don't know what that means in terms we can measure, such as number of employees).

5 ENTER VDT AND VITEPROJECT

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 in order to try to understand and estimate the work of engineering design teams. The Stanford researchers found nothing they could use in the project management literature, nothing that explained the variation in team performance they had subjectively observed. They turned to the literature on organizations and resonated with contingency theory and its information processing view. The challenge then 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 (VDT) [6], to be used in educational settings, and then a commercial product, ViteProject [7]. The displays presented here are from ViteProject.

The translation from theory to simulation is no small feat because of the granularity needed for simulation, which granularity is nearly always absent from theory and even experiments that test such theories. The core of CMOT is the translation process, a subject beyond the scope of this paper.

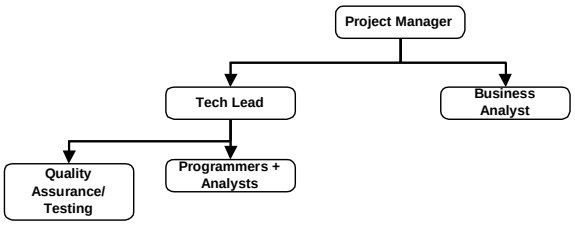
A few details of the translation may suffice to give the reader a sense of the magnitude of the task.

1. To model "change propagation" or "failure dependence." When projects are "fast-tracked," that is, composed of several parallel, simultaneous activities, information or defects found in down-stream activities needs to be communicated to other (possibly still up-stream) dependent tasks. In other words, the simulator needs to make work that is not on the original precedence diagram. The solution was to create activities composed of sub-activities. The simulator assumes there are 20 sub-activities per activity, and that each sub-activity takes 1/20 of the total effort. At the end of each sub-activity the simulator checks to see if it's possible that an exception occurred. This is based on a percentage entered by the user/modeler. If an exception has occurred, then 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.
2. To model centralization. Centralization refers to the degree to which exceptions are handled centrally vs. locally. If exceptions are handled centrally, then often there is work not shown on the precedence diagram to communicate and propagate the exception conditions upwards 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, then it is possible that the actors up the hierarchy get backlogged and cannot respond in time, a delegation by default. Also, central decision-makers 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 in hierarchy of the decision-maker.
3. To model 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." [7, p. 3-34] 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.

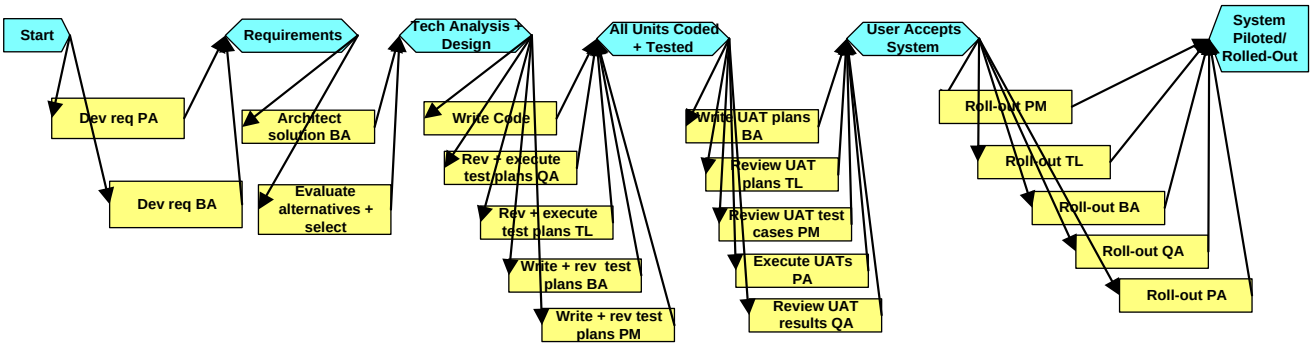
6 EXAMPLE

Baseline

A concrete example may be the best way to illustrate the translation from theory to simulator. Imagine a simple, though realistic, project with an exception hierarchy of:



And further imagine a precedence diagram of activities:

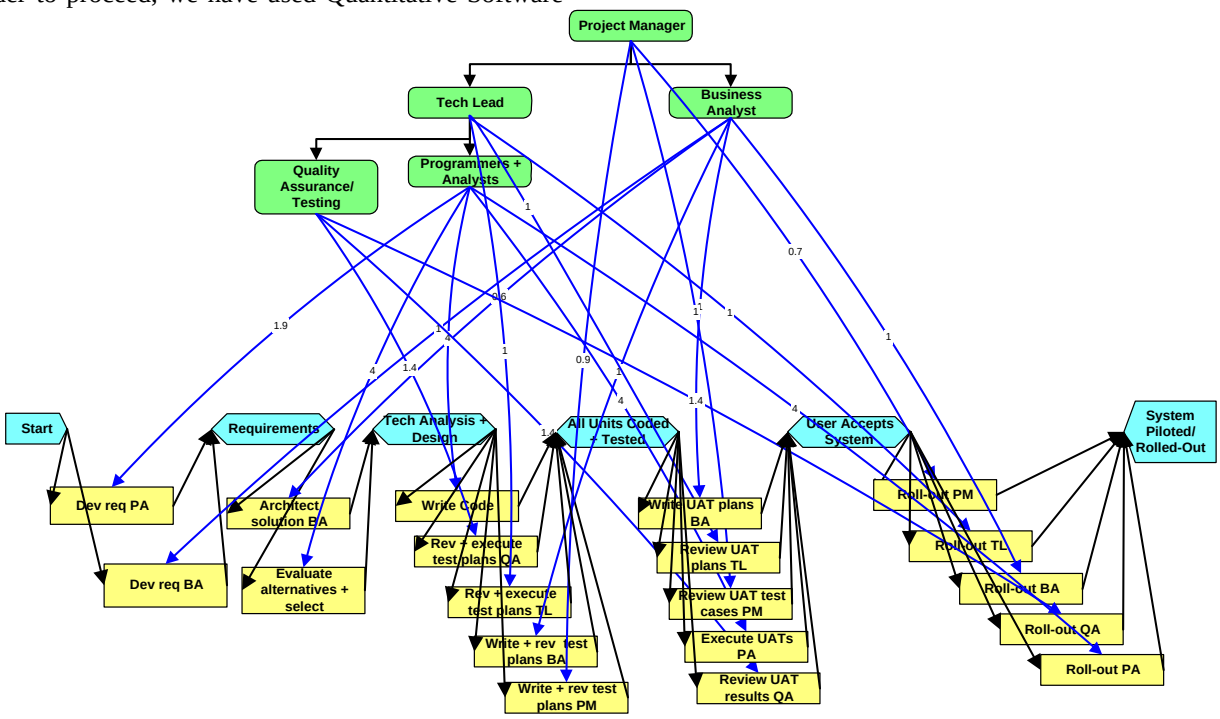


It is an excerpt of a standard software development project. The milestones run along the top and the activities to achieve each milestone are performed in parallel between each milestone. This example assumes that 50,000 lines of code need to developed to deliver specific functionality for a business 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 is available for the project and the rate at which they can be absorbed.

Our job is to optimize the fit between the organization structure and the work to be accomplished.

In order to proceed, we have used Quantitative Software Management's SLIM-Estimate macro software project estimation tool [8] to estimate the overall duration, the duration and effort for each milestone, and the peak number of staff required for each milestone. We have than allocated the effort between milestones to individual job types in accordance with the real project that this one mirrors. For example. There are four people in the "Programmers + Analysts" box and 1.4 full-time equivalents in the "Quality Assurance/Testing" one.

Our next step is to connect a responsible party to each activity:



The barely-readable figures on each arc from an actor to an activity is the number of full-time equivalent actors assigned to activity. In accordance with the actual project, we have a pool of actors, of which only a subset are assigned to a particular activity at any one time.

Next we use VitеProject to simulate the work flow implied by this configuration of actors and activities as a way to baseline our model and compare it with the SLIM model.

Results

The next steps are divided into two activities: calibration and then optimization. Calibration is getting the VitеProject description and results to match SLIM's. Once that is done we try to reduce the duration by changing those variables that we can, and we use information VitеProject gives us to do that.

Here are the VitеProject results obtained in this case:

Scenario	CPM Duration	Sim. Duration	CPM Cost	Sim. Cost	Description
1	386 days	380 days	605.4K	605.4K	Baseline
2	455 days	450 days	553.8K	553.8K	Centralization & formalization low; experience high & 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 & 20% rate
6	338 days	445 days	384.8K	568.6K	Double pre-release failure rate
7	290 days	384 days	323.4K	479.6K	Increased skill of programmers for 2X pre-release 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 QA, + 1 PA
10	280 days	313 days	384.8K	486.3K	+ 1 PA during most intense coding
11	278 days	306 days	378.3K	480.9K	Increased TL skill & pay

The first five scenarios capture the calibration and the rest the optimization.

One can make several observations at once:

1. The duration predicted by traditional project management tools (e.g., Microsoft Project), that is, those that calculate the critical path (the CPM Duration column in the table) is inaccurate with respect to capturing what really happens on a project.
2. SLIM estimated that the project would take 386 days at a cost of US$605.4K. Using VitеProject we have reduced them to 306 days and US$480.9K. These represent savings of 26% in duration and in cost. Clearly, one can save duration AND cost!

The rest of the paper explains the calibration and optimization steps, all guided by displays generated by VitеProject.

Calibration

SLIM, like any other macro estimation tool, builds in some of the variables that VitеProject simulates, such as error rates between activities, communications overhead, the effects of experience in role, and the effects of centralization and formalization. We must, therefore, factor those variables out of the SLIM estimate and make them explicit in our VitеProject model.

The first change is to reflect 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. Typically in software projects and organizations exception handling is responded to locally, not centrally. The trade-off here is that central exception handling usually yields higher quality because of a more global view, but takes longer.

Formalization characterizes communication: is via formal, written memos, or via less formal hallway conversations, for example. Again, typically in software projects and organizations formalization is low. The trade-off is similar to that for centralization: low formalization means that not everything that needs to be known will be communicated up and therefore, especially in parallel tasks, there will be knowledge that needs to be known in one fork that is not communicated there and that will require rework when it does become known. The purpose of more formal communication is make information more globally available – at the expense of speed.

Both SLIM and VitеProject use a default experience level. For this particular project the project leader has high experience and the quality assurance team has low experience. This is based on the real project being modeled.

Those adjustments comprise Scenario 2, above. Scenario 3 models the effects of one type of error. There are two kinds of errors modeled in VitеProject:

- Functional errors – Made by each actor working on a specific activity that is caught during that activity and may extend it, but no other activity.
- Project errors – The error is caught late in the cycle and causes rework to previous, upstream activities.

Scenario 3 models the effect of a 20% rework rate applied to an activity. The 20% figure reflects actual project expe-

rience in this case. It means that one in five sub-tasks internal to an activity have to be reworked. The identification of such errors might be from inspections or testing.

Scenario 4 reflects the observation that adding the effects of centralization, formalization, experience, and functional errors has increased the duration 1/3. This can be illustrated below, in the figure for Scenario 6. Note that each horizontal histogram represents an activity and is composed of four components: work, rework, coordination, and waiting time. SLIM represents this 4-tuple as a single item: work. Our task is to tease apart the SLIM work unit into its four constituents. The increased duration is the effect, so we reduce the effort by 1/3 to get back to the baseline estimate of SLIM. That is, we have set the VitеProject work to be 2/3 of the SLIM estimate and the sum of rework, coordination, and waiting time is the remaining 1/3.

The SLIM estimate was 386 days and VitеProject now obtains a simulated duration of 381, close enough for real world application.

Scenario 5 is our final attempt to get close to the original SLIM estimate by adding the effect of project errors at the rate of 20% for those tasks likely to require extensive rework. This brings the simulated duration to 383 days vs. SLIM's 386 days. The three-day difference is insignificant for our purposes.

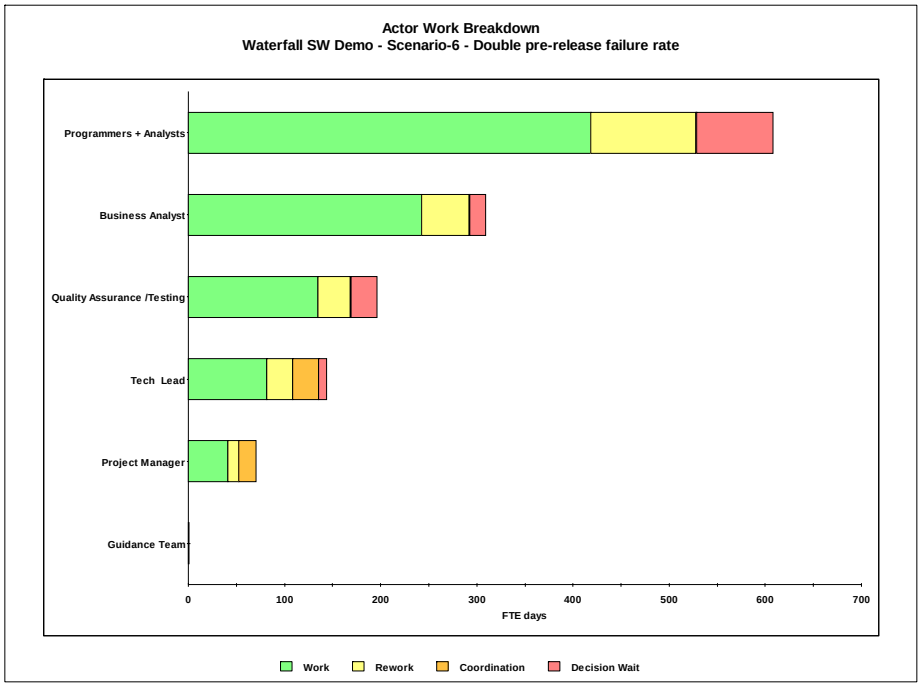
What if questions

Before we entertain optimization we examine a question that the project leader has asked us: what would be the effect if the pre-release error rate were double the typical and what would be a reasonable antidote? The reason the PL asked this question is simply 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%. Roughly, the project will slip about 20% of its schedule and effort.

Where to begin searching for an antidote?

If one looked no further than which type of actor consumed the greatest duration, then that might be a start. As the figure below 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 high within the same pay scale? This would be equivalent to selecting programmers in the pay range with the greatest process maturity, for example.



Scenario 7 increases PA experience from the default of medium to high and the result is to get back virtually all of the loss of a 2X pre-release failure rate.

Optimization

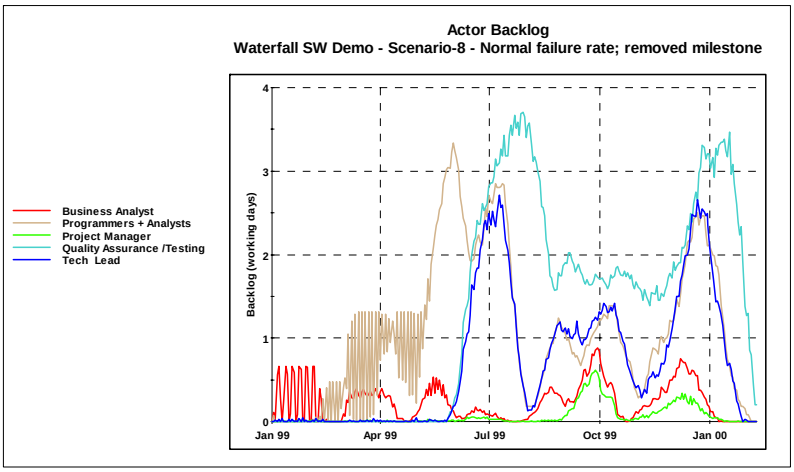
We now enter the more interesting phase: how to reduce duration and costs by improving efficiency and quality.

If structure and work are inter-related, what would be the effect of restructuring the work? To illustrate a possibility, we removed the integration test milestone because it synchronizes according to the slowest completion duration and we can think of a way that once some of the actors have completed their integration test tasks they

could go directly to system test. For example, those writing test cases do not have to wait for a synchronizing activity like completion of a milestone to proceed to working on the next milestone.

The effect of removing the integration test milestone is none because we are still waiting for the slowest performers! The overlap we might have achieved does not materialize because the actors with the longest duration during integration testing are also those with the longest duration of the next activity, namely system testing.

In the next scenario we look at communication backlogs because they so profoundly affect quality and quality reverberates throughout a project.



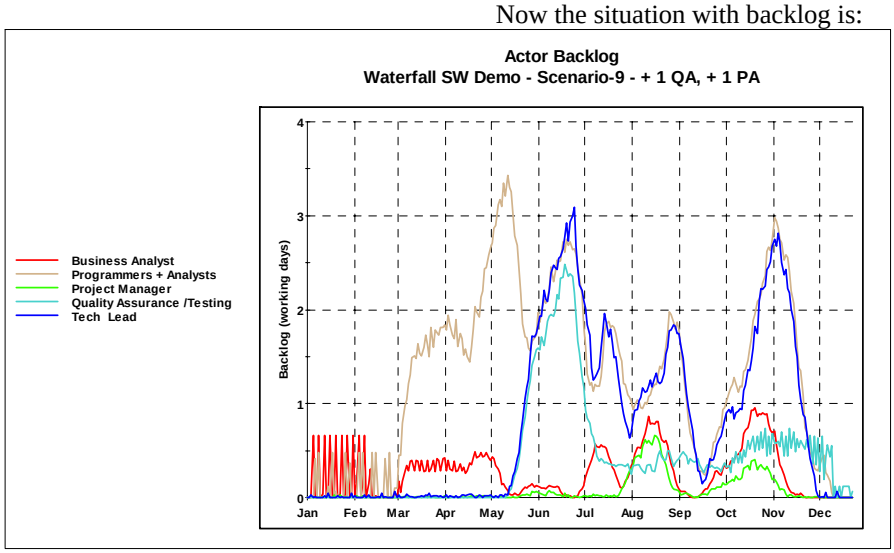
The figure indicates that PAs and quality assurance analysts are missing 3-4 days of messages during the middle of the project (centered around July). It would be natural to add a PA and QA FTE during those peak periods to see the effect.

Adding such resources reflects two fundamental assumptions that are realistic for this modeled projects:

1. Resources are available for relatively short durations (presumably from a pool).
2. 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.

The result of these small additions are reflected in Scenario 9: the duration reduces about 50 days and the cost only goes up about US$2-3K. Clearly, we wish we could buy 50 days of schedule for US$3,000!

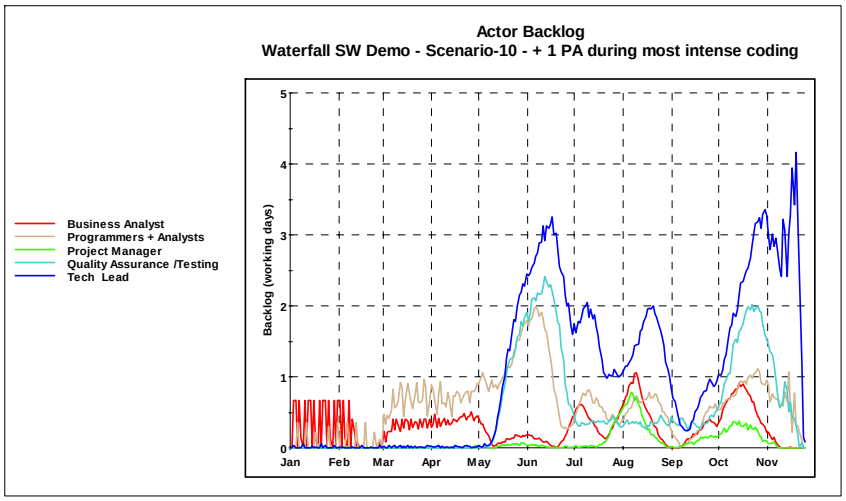
Now the situation with backlog is:



It looks like programmer/analysts are still backlogged around the second week in May. We look at the Gantt chart and see that that is during coding, so we add another programmer to that period alone.

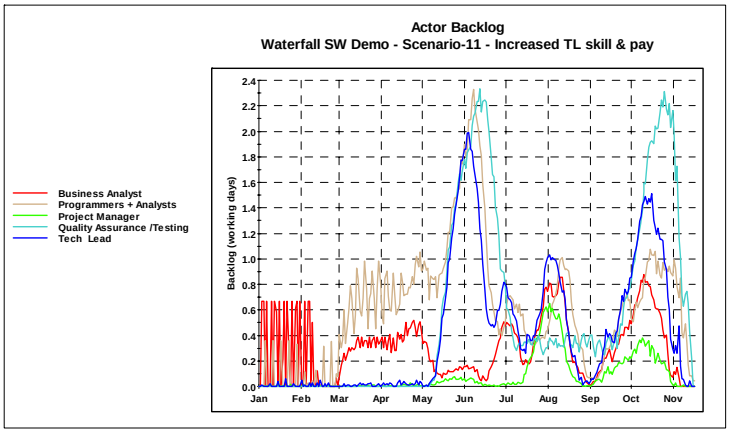
The result is Scenario 10, where we have gained another 20 days of schedule for about US$1K increase in cost.

The backlog situation now stands at:



Clearly the technical lead is backlogged now. We increase his skill level to see its effect, which is Scenario 11. This scenario saves another seven days of duration and also saves about US$5K because the tech lead's subordinates do not have to wait so long for his decisions.

The situation with backlog now is:



While there is still room for improvement, the backlogs are pretty even and they are acceptable to the project team.

7 CONCLUSION

ViteProject opens the door to a level of specificity never before possible. It is a member of emerging tools in the category of computational and mathematical organization theory. It is based on a solid foundation of theory (contingency theory and the view of the organization as an information processing entity) and has proven itself in field trials on real software projects.

ACKNOWLEDGEMENTS

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

REFERENCES

1. Burton, Richard M., & Obel, Børge. (1998). *Strategic organizational diagnosis and design: developing theory for application* (2nd ed.). Boston: Kluwer.
2. Carley, Kathleen, & Prietula, Michael. (Eds.). (1994). *Computational organization theory*. Hillsdale, NJ: Lawrence Erlbaum.
3. *Computational & Mathematical Organization Theory*. Norwell, MA: Kluwer. [Four issues per year, begun in October, 1995, ISSN 1381-298X]
4. Galbraith, Jay. (1974). Organization design: an information processing view. *Interfaces*, *4*(3), 28-36.
5. Gersick, Connie. (1988). Time and transition in work teams: toward a new model of group development. *Academy of Management Journal*, *31*(1), 9-41.
6. Jin, Yan, & Levitt, Raymond. (Fall, 1996). The virtual design team: a computational model of project organizations. *Computational & Mathematical Organization Theory*, *2*(3), 171-196.

7. Levitt, Raymond. (1998). *The ViteProject handbook: a user's guide to modeling and analyzing project work processes and organizations*. Palo Alto: Vité.

8. Putnam, Lawrence H., & Myers, Ware. (1992). *Measures for excellence: reliable software on time, within budget*. Englewood Cliffs: Prentice-Hall.