A Model of Correlated Team Behavior in a Software Development Environment

Thomas E. Potok/Oak Ridge National Laboratory
Mladen A. Vouk/North Carolina State University

Abstract

In today's highly competitive software development environments, accurately estimating software duration and cost can often mean the difference between project success or failure. Traditional software development estimation techniques often assume that software development teams operate independently from task to task. This assumption allows task estimate covariances to be ignored. However, there is ample evidence that the behavior of a software development team is not independent, but is consistent over the life of a project. This means that the accuracy of software estimates may suffer due to this simplifying assumption. We present results from what we believe is the first general model for representing correlated team behavior in a software development environment. This model shows that strong teams that behave in a correlated manor out perform teams that operate randomly. What is not as obvious is that weak teams that consistently perform poorly have a lower overall productivity than a random team. This suggests that the best cure for a poorly performing team may be to randomly shuffle the team members.

1. Introduction

As software development processes mature, and software development cycles shorten there is becoming less and less margin for error in the successful development of a software project. Where a few years ago developer's estimates were routinely doubled before a schedule commitment was made, now it is not uncommon for a developer's best estimates to actually be shortened before a project commitment is made.

1.1. Related work

The most frequently used software estimation models are typically activity based models. These models organize a software project into a network of several smaller parts, commonly called tasks. Estimates of duration are made for each task, then the overall project duration is computed from task information over for the entire project network. Often an estimate of the duration for a task is based on a range a values, or an approximate PDF for the duration of each task. This implies that the overall project duration can be estimated by the joint PDF of the duration of the relevant tasks. Clearly deriving or simulating this joint distribution can be a very difficult task, particularly if the tasks are correlated in some way. However, the tasks are typically assumed to be independent. That is the task duration times are 1) independent and identically distributed, and 2) the mean and variance of each duration time is known. If this assumption holds, then estimating the mean and variance of the joint PDF can be a fairly trivial task. The PERT model is probably the best-known activity based model used for software development [1]. Other representative activity model include: Markov chains [2]; and PETAN networks combining general activity networks and Petri nets [3]. There has also been work on simulating the activity with the software development process, Potok and Vouk use common business model effects to derive a distribution for software task completion based on enterprise, methodology, and process factors [4].

The key issues related to modeling the behavior of software develop teams are, 1) is it valid to assume that software development teams act independently from task to task in a project, and 2) how can the correlated behavior of a software development team be effectively modeled.

We propose a simulation model that supports correlated team behavior, i.e., a strong team will exhibit high-productivity over all the tasks, while a weak team will perform with low productivity over all the tasks. The correlated team model uses a preset factor to estimate the team proficiency, then uses randomly selected values from tailored distributions that produce correlated samples. These samples are then used to determine the task and project duration for a team behaving in a correlated manor. We believe that this approach can provide far greater insight into both estimating and understanding how software development teams operate.

2. Background

We now describe a general model that we have used to estimate task and project duration for software development [4]. In the context of this paper we define average productivity of a software professional in thousands of LOC (KLOC) per person-month, but with an understanding that the effort (or time) expended includes many non-coding activities that are necessary in developing a viable project. A software team may consist of one or more software professionals, not all of which need to be engaged in software coding and testing activities. We will express software team productivity in terms of KLOC per calendar month.

A task or activity is a unit of work that produces a subcomponent of the overall project. Work performed in a task is well defined with a planned start and finish time. Tasks can be viewed as individual segments of a project starting at the completion of the previous task, which we call individual tasks.

1.2. Model

We assume that the distribution of task durations absent of external influences is uniform and bounded by the minimum and maximum task duration ranges (TDRs) for the task.

The granularity of our model is at the level of project tasks. Therefore, in addition to individual tasks, we recognize aggregate tasks. The start of an aggregate task is conditioned on completion of the task that precedes it. We represent the duration of a project task as a function of team productivity requiring estimation of the effective size or complexity of a project task (e.g., in terms equivalent KLOC), and of the average team productivity over the task in the same units (e.g., in KLOC/(calendar development month). The duration of a task is then

$$\text{Task Duration} = \frac{\text{Task Size}}{\text{Team Productivity}}. \qquad (1)$$

The relationship between task duration and size is linear if and only if team productivity is constant as the size of a project increases. Next we define the metrics used to describe planned and actual task duration. The minimum and maximum team productivity range is used to estimate the minimum and maximum task duration range (TDR).

Let t be the duration of a task. For each individual task j, within project i, four metrics are recorded: $t_{i,j,act}$, the actual task duration time; $t_{i,j,plan}$, the planned duration time; $t_{i,j,max}$, the maximum duration time; and $t_{i,j,min}$, the minimum duration period. The cumulative task completion time up to and including task j, for a given project i, is a function of the preceding sequence of tasks, i.e., $d_{i,j,act} = \sum_{u=1}^{j} t_{i,u,act}$. For example, $d_{1,1,plan}=t_{1,1,plan}$, $d_{1,2,plan}=t_{1,1,plan}+t_{1,2,plan}$, etc. As with the individual tasks, there are four cumulative durations: actual, planned, maximum, and minimum.

The minimum and maximum duration times for a task define a range of possible completion times for that task. This duration is represented for task j with the random variable T_j. T_j can assume the values between the minimum duration time $t_{i,j,min}$, and the maximum duration $t_{i,j,max}$. Duration of a project with n tasks is a random variable D defined by

$$D=T_1+T_2+...+T_n, \qquad (2)$$

Where T_j is a discrete independent random variable within the task duration range of the critical path, and D is a discrete random variable representing the project duration.

To complete the basic model, we assume that the software development team operates independently from task to task, then to simulate the duration of a project whose tasks fall within the intervals $[t_{i,min}, t_{i,max}]$, $i=1,...,n$, we take a sample from each interval according to the distribution assigned to that interval. This provides an estimate of the individual task duration times for the project. From this estimate, the aggregate durations can be determined, as well as the overall project duration time. We repeat this sampling until the required simulation accuracy is achieved. We will now quantify the sampling distribution that produces correlated team behavior.

3. Correlated team extension

If we no longer assume that the tasks are independent, but correlated. We now need a method for obtaining random correlated samples. For example, if the correlation level is set to 75%, then all simulated projects will see task durations that correlated to approximately 75%.

The correlated task duration samples for a project form the random column vector $\mathbf{T}\equiv[T_1,...,T_n]^T$, where T_j is a random variable representing the duration of a task j in project i which can assume values between the minimum duration time $t_{i,j,min}$, and the maximum duration $t_{i,j,max}$.

From **T,** we can then define the multivariate distribution with the mean vector

$$\mu_{\mathbf{T}} \equiv [E(T_1), E(T_2), ..., E(T_n)] \quad (3)$$

With the variance-covariance matrix taking the form

$$\Sigma_{\mathbf{X}} \equiv \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2n} \\ \cdots & \cdots & \ddots & \cdots \\ \sigma_{n1} & \sigma_{n2} & & \sigma_n^2 \end{bmatrix}. \quad (4)$$

Due to the very limited study in this area we make the assumption that there is a common correlation between the team productivity rates of any two tasks. This implies that a software development team will produce software at a correlated level for any two tasks. Translating team productivity to task duration (see Equation (1)) this correlation is seen in the relative location within T_j a duration sample is collected. This matrix can be expressed in terms of correlations, i.e., $\rho_{\mathbf{XY}} = \sigma_{\mathbf{XY}} / \sigma_{\mathbf{X}}\sigma_{\mathbf{Y}}$ which produces $\Sigma_{\mathbf{T}}$ that has the "repeated-measures" form

$$\Sigma_{\mathbf{T}} \equiv \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 & \cdots & \rho\sigma_1\sigma_n \\ \rho\sigma_2\sigma_1 & \sigma_2^2 & \cdots & \rho\sigma_2\sigma_n \\ \cdots & \cdots & \ddots & \cdots \\ \rho\sigma_n\sigma_1 & \rho\sigma_n\sigma_2 & & \sigma_n^2 \end{bmatrix}. \quad (5)$$

The method used to sample **T** is approximate; but if the univariate marginal distributions of **T** are symmetric, then the approximation is excellent in the sense that the generated random vector has exactly the target mean, Equation (3), while the target covariance matrix, Equation (5), is a very close approximation.

4. Analysis

We now used the simulation model described above to explore the impact of correlated and independent teams on the productivity of a hypothetical project. For this analysis, we define a project that has five equally sized tasks. For simplicity, the planned duration for each task is set to 10 weeks. This translates to planned deadlines at 10, 20, 30, 40 and 50 weeks, respectively. We also assume that an equivalent of 10 KLOC is developed during each task. We further assume that the development team productivity[1] for the project operates within the ranges of 500 LOC/week to 1250 LOC/week. From equation (1) it follows that each project task has a duration range between 8 to 20 weeks. For this project we run two simulations, the first assumes that a team works independently from task to task, and the second assumes that the team behavior is correlated at a 75% level from task to task.

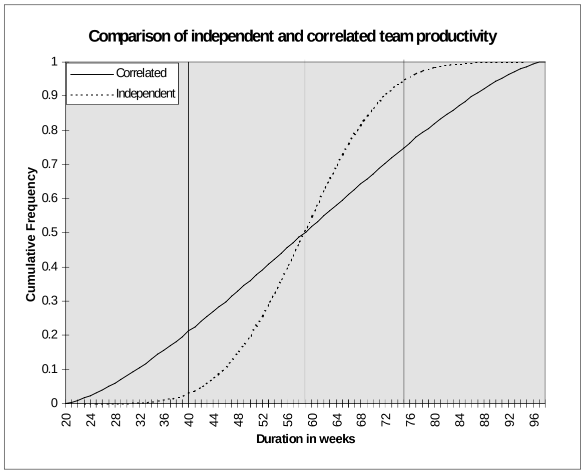

Figure 1 A comparison of correlated and random team behavior

Figure 1 shows the cumulative distributions for the duration of the project developed by a random team (dashed line) and a correlated team (solid line). Not surprisingly, a development team that consistently performs with high-productivity will have a higher likelihood early completion of a project than a team that exhibits independent or random behavior. For example, at a 40-week deadline, the correlated team has roughly a 20% chance of success, while a random team has about a 2% chance. Conversely, at a 75-week deadline, a random team clearly out performs a poor performing team.

It appears that a correlated team does indeed perform quite differently than a random team, however, it is surprising that a strongly correlated team that has consistently weak performance will take longer on average than a random team. Based on this result, it appears that a team that consistently performs poorly may benefit through randomization in order to see improved performance.

Figure 2 shows the typical effect of the business model on correlated and random team behavior. Again the deadline is set to 50 weeks, but it is now strongly enforced, meaning that there are strong

[1] It is also assumed that the average team size is around 10 software professionals.

negative consequences if the deadline is not met. This plot shows that correlated teams have a smaller variance around a task deadline than do teams that exhibit random behavior from task to task. This smaller variance shows as expected that the correlated team performs more consistently than the random team, however, may have less flexibility to significantly improve its behavior during the project life-cycle. From these plots, it is clear that the correlated effects of the software development team can alter the results of project estimates that are based on independent team behavior.

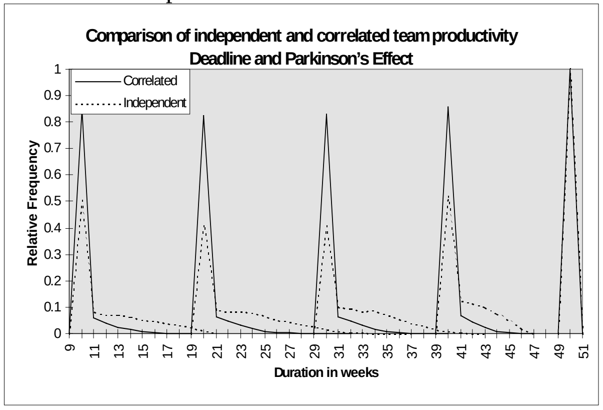

Figure 2 A comparison of correlated and random team behavior both under common business effects.

5. Summary

We present a model to support correlated team behavior over a software development project. Rather than a team performing strongly on one task, then poorly on the next, a team can be simulated to perform consistently over all tasks. Results from this model sustain the contention that software development teams do not perform independently from task to task, and that assuming so can limit the accuracy of a software project estimate.

The results further indicate the obvious, that good teams perform better than random teams, and not as clear, that random teams perform better than poor teams. However, it also provides some insight into managing such teams. Clearly a good team should be maintained and used where high productivity is needed. For a weak team, it seems they should be used when low productivity is acceptable, or the team should be randomized in some way to improve their productivity.

Further investigation is required to understand what factors may influence a software development team behavior.

6. Acknowledgments

Oak Ridge National Laboratory, managed by the Lockheed Martin Energy Research Corporation for the U. S. Department of Energy, under contract number DE-AC05-96OR22464. This submitted manuscript has been authored by a contractor of the U. S. Government under Contract No. DE-AC05-96OR22464. Accordingly, the U. S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U. S. Government purposes.

7. References

1. K. R. MacCrimmon and C. A. Ryavec. "An Analytic Study of the PERT Assumptions," *Journal of the Operations Research Society of America*,1964.
2. D. Raffo. "Evaluating the Impact of Process Improvements Quantitatively using Process Modeling," *Proceedings of CASCON'93*, 290-313, 1993.
3. S. E. Elmaghraby, E. I. Baxter and M. A. Vouk. "An Approach to the Modeling and Analysis of Software Production Process," *International Transactions in Operational Research*, 2(1): 117-135, 1995.
4. T. E. Potok and M. A. Vouk. "The Effects of the Business Model on Object-Oriented Software Development Productivity," submitted for publication to *IBM Systems Journal*, 1996.