Enhancing the Cocomo Estimation Models

Team task assignments appear to have a significant potential impact on a project's overall success. In this article, the authors propose task assignment effort adjustment factors that can help tune existing estimation models. They show significant improvements in the predictive abilities of both Cocomo I and II by enhancing them with these factors.

Joanne Hale, Allen Parrish, and Brandon Dixon, *University of Alabama*
Randy K. Smith, *Jacksonville State University*

It is a well-accepted project management axiom that increasing the number of people working concurrently on a project does not necessarily result in a corresponding increase in productivity (Brooks' Law says, "Adding more programmers to a late project makes it later").[1] In a similar vein, there is an expectation that having individuals focused on a single task is an important mechanism to foster productivity.[2] We refer to the manner in which tasks are shared among team members as a *task assignment pattern.*

Although certain task assignment patterns are widely cited as desirable in the software engineering literature, there are remarkably few supporting empirical studies. In previous work, we reported on an empirical study in this area, identifying three new metrics to measure various dimensions of a given task assignment pattern.[3]

We called these metrics *intensity, concurrency*, and *fragmentation*. Intensity measures the degree of focus on a particular task. A task with a high level of intensity is worked on regularly, while tasks with low levels of intensity have frequent or long breaks in the work activity (perhaps to let work be completed on other tasks). Concurrency refers to the degree to which team members are working together cooperatively on given tasks. Fragmentation measures the degree to which a team's time is fragmented across multiple tasks. We discovered that these factors substantially impact development effort and that they significantly improve the performance of the intermediate Cocomo I estimation model.[4,5]

Derived from our previous work, we now propose a simple adjustment process for scaling the results of existing effort estimation models to reflect the impact of our task assignment metrics (TAMs). Because it is a simple multiplicative scaling process, we can apply it to the results of an existing estimation model. Consequently, we can adjust existing models (which do not take into account task assignment patterns) to reflect what researchers have already found to impact development effort.

0740-7459/00/$10.00 © 2000 IEEE

Computing the Task Assignment Metrics

We proposed methods for computing intensity, concurrency, and fragmentation in our previous study.[1] All three factors begin with the notion of a *time unit*, which is just an interval of time of fixed duration. We chose time units to have duration of one day, although other time unit sizes could be used (with different resulting values for intensity, concurrency, and fragmentation).

As an example, consider a task that is eight days (time units) in duration. Suppose that work is reported on this task by at least one developer on five of the eight days; for three of the eight days, no one reported work on the task—thus, the intensity associated with this particular task is 5/8.

With regard to concurrency, suppose that the number of developers reporting work on the task on each of the five days is given by the set {3,2,3,2,3}. Concurrency is computed as an average over the five days—thus, (3+2+3+2+3)/5.

With regard to fragmentation, the computation is slightly more complex. The table below represents the number of tasks for which each of the developers reported activity during each of time units when work was reported for task *j*. That is, t_1, t_3, t_4, t_7 and t_8 are time units when someone reported work on task *j*. For example, as the table shows during time unit t_1, developer p_1 reported work on three different tasks, developer p_5 reported work on two different tasks, and developer p_9 reported work on one task.

Task *j*	t_1	t_3	t_4	t_7	t_8
p_1	3		2	1	1
p_2		2	1	1	
p_5	2		1		1
p_9	1	2			2

We compute fragmentation for this particular task *j* by averaging the number of tasks each developer reports over the duration of task *j*. In particular, the sum is taken for each row (3+2+1+1, 2+1+1, 2+1+1, 1+2+2) and then divided over the number of team members (4).

Reference

1. R. Smith, J. Hale, and A. Parrish, "An Empirical Study using Task Assignment Patterns to Improve the Accuracy of Software Effort Estimation," to be published in *IEEE Trans. Software Eng.*

Task Assignment Metrics

Formal definitions of our three TAMs appear elsewhere.[3] Here, we provide the basic substance of those definitions, while the sidebar "Computing the Task Assignment Metrics" shows how to compute values for intensity, concurrency, and fragmentation.

We assume that a *task* is some separately identifiable activity, which could include a software module, application, or a particular development activity. Although characterizing individual activities into discrete, independent tasks might be difficult in some cases, we require the project manager to construct such a characterization for our factors to be quantitatively measured.

A key underlying concept in all three definitions is the notion of a *time unit*, which is a discrete interval of time of some fixed duration. A task is characterized by its *duration* (the number of time units between when work on a task begins and when it is completed). We define our TAMs with respect to tasks—they are properties of tasks, rather than of individuals. This lets us use the factors to scale the effort estimate associated with particular tasks.

Intensity (for task *j*) is then defined as the number of time units devoted to *j* divided by the total duration of *j*.

A task with a high intensity level is worked on with sharp focus and few or no hiatuses, while a low intensity level is associated with a task that might have remained untouched for long periods of time.

Concurrency (for task *j* at time unit *x*) is defined as the number of developers working on task *j* during time unit *x*. To obtain the concurrency for task *j* over its entire duration, we average the concurrency for all time units when work was reported on *j*. Although concurrency does not directly measure interactions among team members, we expect a strong relationship between interactions and the degree to which team members report simultaneous activity on the task.

Fragmentation measures the degree to which a team's time is fragmented over multiple tasks. To determine the fragmentation associated with task *j*, we determine the number of tasks each developer reports during each time unit when work is reported on *j*. We compute the sum of all such reported tasks for each developer, and then we compute a team average to determine the fragmentation for task *j*. Thus, fragmentation is computed as the total number of reported tasks for all developers divided by the number of developers reporting work on *j*.

Research Study

The overall project we studied was a client–server application for a state government agency, supporting in excess of 400 users distributed through a statewide network and encompassing agency accounting, procurement, inventory control, and personnel. An external contractor with a 16-person development team undertook the project. The external contractor constructed

Table 1

Performance of Task Assignment Metrics

Metric	Calibrated intermediate Cocomo I	Calibrated Cocomo I, augmented with three TAMs	Cocomo II post architecture	Cocomo II, augmented with three TAMs
Average square-root error (ASRE)	7.2	6.1	7.0	5.8
PRED(30%)	36%	49%	49%	60%

the system's 160 modules (the object of analysis in this study) and successfully completed user testing and acceptance. We obtained data for the Cocomo models from Hierarchical-Input-Process-Output (HIPO) diagrams, and we obtained the TAMs from the time accounting system the contractor used for billing.

Within this studied environment, we found the following:

- Development effort decreased as intensity increased. This demonstrated impact of intensity supports Tom DeMarco and Timothy Lister's work;[2] they discovered in their survey that uninterrupted time dramatically increases productivity. High productivity requires single-minded work time; thus, every interruption costs additional immersion time before the developer can again single-mindedly focus on the task at hand.
- Development effort increased with concurrency (degree of team collaboration). Thus, teams took less time to complete a task when members could work more independently. This might be due to a lack of team discipline, as suggested in Vic Basili and Robert Reiter's work,[6] or to a natural tendency of developers to prefer working alone, as Dan Cougar and Heimo Adelsberger suggested.[7]
- Development effort increased with fragmentation. As a developer's time is divided over greater numbers of unrelated activities, he or she can spend less time on productive work because more time must be spent shifting between tasks.

Having found that the three TAMs significantly impact development effort, we turned our attention to improving the widely used Cocomo estimation models. In our previous study and in subsequent analysis, we examined four models relevant here: an intermediate Cocomo I model tuned to the specific development environment, the Cocomo II post architecture model using the 1997 calibrated parameters, and two additional models augmenting Cocomo I and II with the three TAMs. The TAM models use a natural logarithm transformation to linearize the original multiplicative model and linear regression to estimate the TAM coefficients.

To compare the four models, we used two metrics:

- the average square-root error (ASRE),[8] measuring the ability of the model to fit the data from which it was derived (in a perfect prediction the ASRE approaches 0); and
- the percentage of estimates having a magnitude of relative error9 of 30% or less, measuring the ability of the model to predict future development effort. (The magnitude of relative error is the size of the deviation between the predicted effort and the actual effort for a particular task *t*. Predicted effort is calculated with task *t* removed from the data set. The notation PRED(X%) refers to the percentage of task estimates having a deviation less than X%.)

As Table 1 shows, both metrics reveal a significant improvement provided by the three TAMs.

Adjusting Effort Estimation Models

We believe that our three TAMs have the potential to significantly enhance the predictive power of the commonly used Cocomo effort estimation models. However, the usability of such an approach is limited. It would require an estimate of each of these three factors in addition to the large number of factors already collected for a firm's standard effort estimation model. This data collection is complicated because, in the previously examined models, all three metrics are represented by ratio data rather than the nominal data used in Cocomo models, such

Table 2

Proposed Task Assignment Effort Adjustment Factors

Metric	Effort Adjustment Factor Very low	Typical	Very high
Intensity	1.3	1	0.7
Concurrency	0.9	1	1.3
Fragmentation	0.9	1	1.4

as development flexibility ranging from *very low* to *extra high*.

The usability of the TAMs depends not only on their power to improve effort estimation but also on their simplicity. In this light, we propose a set of simple multiplicative effort adjustment factors that can augment the power of an effort estimation model with the TAMs. Although some predictive power is lost in the shift from ratio to nominal data, the resulting ease of application is tremendously attractive.

The basic steps needed to augment an effort estimation model are as follows:

1. Over a period of time, collect data regarding the levels of intensity, concurrency, and fragmentation experienced within the organization.
2. Once norms for the three TAMs are established, begin to compare those experienced for a specific development effort to the organizational average. Classify a project believed to fall in the lower quartile for a metric as *very low*. Classify a project believed to fall in the upper quartile for a metric as *very high*. Consider other projects to be *typical* with respect to that metric.
3. Apply each of the three multiplicative effort adjustment factors (see Table 2) to the estimation calculated using your standard model (if the USC Cocomo II software is used, take advantage of the available user-defined cost drivers).
4. Monitor the quality of the resulting estimates, and calibrate to your particular development environment.

Obviously, these factors are not independent. As individual team members are asked to juggle greater numbers of tasks, both intensity and fragmentation tend to suffer, magnifying their individual impact. For example, you might expect low intensity to frequently occur in the context of high fragmentation. In such a case, the application of our effort adjustment factors nearly doubles the estimated level of effort. Of course, the tendency in such cases might be to add additional team members, likely increasing the level of concurrency. When all three factors negatively impact the project, the total effort is scaled by nearly two-and-a-half times. On the other hand, when all three factors positively impact the project (high intensity, low concurrency, low fragmentation), the total effort estimation reduces by a factor of nearly one-half.

What Results Should You Expect?

We used the data set from our original study to establish the effort adjustment factors (EAFs) in Table 2. First, we divided the data set in half: set 1 to create the EAFs and set 2 to evaluate their impact on model performance. Second, we determined the relative level of intensity, concurrency, and fragmentation associated with each task in set 1 by comparing the average of each metric within set 1. We then used linear regression (using the necessary log transform) to estimate each multiplicative factor. We evaluated three, four, and five levels of each

Table 3

Impact of Task Assignment Effort Adjustment Factors

Metric	Calibrated Intermediate Cocomo I	Calibrated Cocomo I, augmented with three adjustment factors	Cocomo II post architecture	Cocomo II, augmented with three adjustment factors
Average square root error (ASRE)	8.9	7.7	7.2	6.7
PRED(30%)	29%	35%	37%	43%

metric. We rejected the four- and five-level models because their added complexity failed to result in significant improvement in effort estimation.

Once we determined the EAFs with set 1, we evaluated them with set 2 as in Table 3. We found significant improvement in model performance for both Cocomo I and II. Although individual performance improvements will vary, results of this analysis are promising. Cocomo I prediction accuracy rose from a PRED(30%) of 29% to 35%. Similarly, the prediction accuracy of Cocomo II climbed from PRED(30%) of 37% to 43%.

Our research has shown that software development effort decreases with breaking work assignments down into tasks that can be accomplished individually (to reduce concurrency), compressing the development schedule of tasks (to increase intensity), and letting teams focus on a small number of tasks (to reduce fragmentation).

These results provide valuable guidance for project managers in their endeavors to reduce system development effort. In addition, these results might help improve the predictive ability of the Cocomo effort estimation models, which fail to take into account these TAMs. Rather than starting anew, we recommend adding multiplicative adjustment factors to reflect the impact of task assignment on development effort. We have shown significant improvement in predictive power by augmenting the popular Cocomo I and II models with EAFs to be applied when intensity, fragmentation, and concurrency vary dramatically from the norm.

References

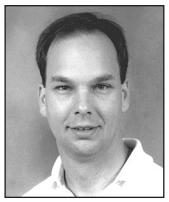
1. F. Brooks, *The Mythical Man-Month*, Addison-Wesley, Reading, Mass., 1975.
2. T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*, Dorset House, New York, 1987.
3. R. Smith, J. Hale, and A. Parrish, "An Empirical Study using Task Assignment Patterns to Improve the Accuracy of Software Effort Estimation," to be published in *IEEE Trans. Software Eng*; contact jhale@cba.ua.edu for a copy.
4. B. Boehm, *Software Engineering Economics*, Prentice-Hall, Upper Saddle River, N.J., 1981.
5. *Cocomo II Model Definition Manual*, Univ. of Southern California Center for Software Eng.; http://sunset.usc.edu/publications/TECHRPTS/index.html (current Oct. 2000).
6. V. Basili and R. Reiter, "An Investigation of Human Factors in Software Development," *Computer*, Vol. 12, No. 12, Dec. 1979, pp. 21–38.
7. J. Cougar and H. Adelsberger, "Comparing Motivation of Programmers and Analysts in Different Soci/Political Environments: Austria Compared to the United States," *Computer Personnel*, Vol. 11, No. 4, Apr. 1988, pp. 13–17.
8. B. Clark, S. Devani-Chulani, and B. Boehm, "Calibrating the Cocomo II Post-Architecture Model," *Proc. 1998 Int'l Conf. Software Eng. Workshop*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1998, pp. 477–480.
9. N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, Brooks/Cole Pub. Co., Stamford, Conn., 1998.

About the Authors

Joanne Hale is an assistant professor in the Department of Information Systems, Statistics, and Management Science at the University of Alabama. Her technical interests include software cost estimation and component-based development and reuse. She received her BS in industrial engineering and her MA in statistics from the University of Missouri, and her PhD in MIS from Texas Tech University. She is a member of the ACM, the IEEE Computer Society, and the Society for Information Management. Contact her at the Dept. of Information Systems, Statistics, and Management Science, Culverhouse College of Commerce and Business Administration, Univ. of Alabama, Tuscaloosa, AL 35487-0226; jhale@cba.ua.edu.

Allen Parrish is an associate professor in the Department of Computer Science at the University of Alabama. His technical interests include component-based software development and data warehousing. He received his PhD in computer and information science from Ohio State University. He is a member of the ACM and the IEEE Computer Society, where he is a member of the Educational Activities Board. Contact him at the Dept. of Computer Science, Univ. of Alabama, Tuscaloosa, AL 35487-0290; parrish@cs.ua.edu.

Randy K. Smith is an assistant professor of computer science at Jacksonville State University. His technical interests include cost-estimation modeling, component-based software development, and distributed software engineering. He received his MS and PhD in computer science from the University of Alabama. He is a member of the IEEE Computer Society, the ACM, and the Association of Information Technology Professionals. Contact him at the Mathematical, Computing, and Information Sciences Dept., 301 Bibb Graves Hall, Jacksonville State Univ., 700 Pelham Rd., North, Jacksonville, AL 36265; rksmith@jsucc.jsu.edu.



Brandon Dixon is an associate professor in the Department of Computer Science at the University of Alabama. His technical interests include computer science theory, algorithm design, and software engineering. He received his PhD in computer science from Princeton University. Contact him at the Dept. of Computer Science, Univ. of Alabama, Tuscaloosa, AL 35487-0290; dixon@cs.ua.edu.