Value Creation and Capture: A Model of the Software Development Process

Todd Little, *Landmark Graphics*

Landmark Graphics supplies software and services to the upstream oil and gas industry. Our software portfolio, which ranges from exploration and drilling to data management and decision analysis, includes more than 60 products consisting of over 50 million lines of source code. For many years, Landmark has been collecting project metrics we wished to harvest to gain insight into key business questions in three areas:

- Optimal release cycle duration (scope/time trade-off)
- Optimal project staffing levels
- Effects of uncertainty

Previous approaches to understanding software dynamics and costs include building systems dynamics models or detailed cost equations.[1,2] Although they can be enlightening, these models are complex, they focus on cost rather than value, and they typically don't consider uncertainty. Other models use net present value to focus on value and uncertainty.[3,4]

Understanding software development dynamics can help an organization maximize value delivery. Using functions and parameters applied in a spreadsheet, this process model facilitates this understanding by examining value creation and value capture in the presence of uncertainty.

My team set out to develop a relatively simple project dynamics model to use in conjunction with market sensitivity and economic analysis to help optimize profitability. Some of our ideas and results are similar to those of Preston Smith and Donald Reinertsen, who examined the impact of time-to-market sensitivity.[5] However, our approach is a more detailed model tuned to software development issues.

The model

Our model features a set of functions and parameters that can be applied in a spreadsheet. You can obtain data curves in many ways, including using industry data, historical metrics, or from the team's subjective assessments.

The curves I present here are of a sample base case in our commercial-project portfolio. Landmark has collected development and testing duration and effort metrics for its projects from 1995 to 2003. Although I believe the curves'

0740-7459/04/$20.00 © 2004 IEEE

shapes are representative of most projects, I don't intend to generalize a specific equation across all projects. I also recognize that software development depends highly on the individuals and teams involved. The model incorporates this variation to the extent that the functions should represent the project and the team involved in the work. Again, I caution against generalizing, yet so long as the limits and assumptions are known, I believe the model provides a basis for solid business decisions.

The parameters and functions that define the essential model are

- *Staff effectiveness:* effective team productivity versus team size
- *Value created:* the software's assumed value versus effective development time
- *Rework time:* time to test and fix the software versus effective development time
- *Value capture:* market delay costs versus time
- *Resources:* team size and cost factors

Each of these functions is a curve representing a given project's performance. Different projects' behaviors can vary depending on factors such as product maturity, team experience, and development process. The base case in this study is a mature product with an existing code base and an established team.

Team size

For our model, we define

$$\textit{Effective team productivity} = \frac{\textit{Team size} \times \textit{Average team productivity}}{\textit{Productivity when team size} = 1}$$

Effective team productivity is dimensionless, while *productivity* can be measured in lines of code or function points.

Frederick Brooks postulated team-productivity diminishing returns as the *n-squared* problem, referring to the n^2 increase in communication channels as you add people to a project.[6] Tarek Abdel-Hamid and Stuart Madnick modeled this with a systems dynamics model and estimated that the communication overhead of a 30-person team would be 50 percent.[1] However, their model estimates that communication overhead goes to 100 percent for a 40-person team, and it predicts a behavior different from that observed with other authors' productivity data. Samuel Conte and his colleagues, using lines of code, observed that *average team productivity* declines exponentially with *team size*.[7] A reformulation of their observations gives

$$\textit{Effective team productivity} = \textit{Team size}^{\alpha},$$

where, for their data, $\alpha = 0.5$. Capers Jones uses function points to provide data including productivity and team size.[8] Using this equation and Jones's data results in an excellent fit with $\alpha = 0.65$, which is the equation we used for this study. Figure 1 shows $\alpha = 0.65$ relative to the curves of Abdel-Hamid, Conte, and Jones's data.

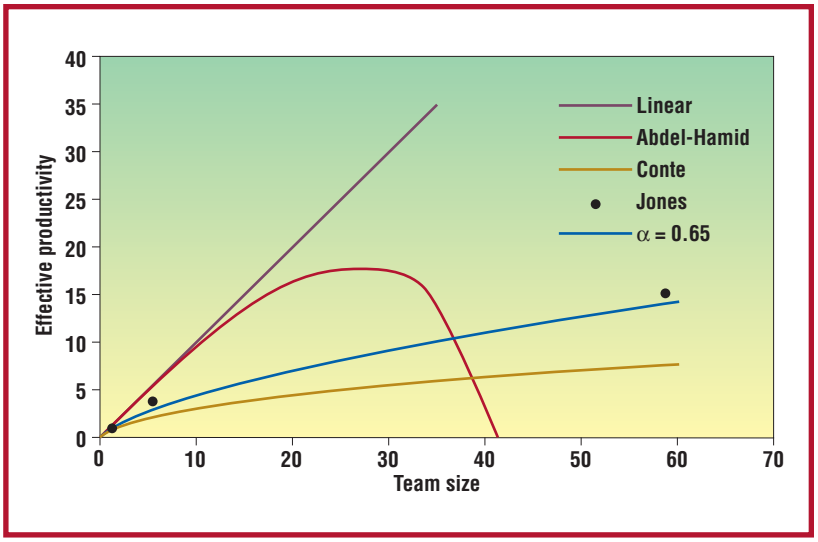


Figure 1. Effective productivity using different models.

Assumed value creation

To assess the developed features' value, we use an approach similar to Minimum Marketable Features—a collection of features required to derive value from the product.[9] As with MMF, we require that each feature set has an associated estimated value and effort cost. We define *value* as the estimated present value of the feature if it were delivered immediately. We give costs in *effective developer days*, similar to how the planning game in *Extreme Programming Explained* uses ideal developer days.[10] We also follow XP's practice of prioritizing features on the basis of value, although we prioritized using a variant of the *profitability index*—the ratio of present value to cost. Alternative prioritization approaches should be compatible with the model.[11,12] The result is the base curve in Figure 2. The curve's discrete nature is a result of the value materializing only at the feature set's completion. The lowest line depicts poorly prioritized features, and the remaining curves represent random prioritization.

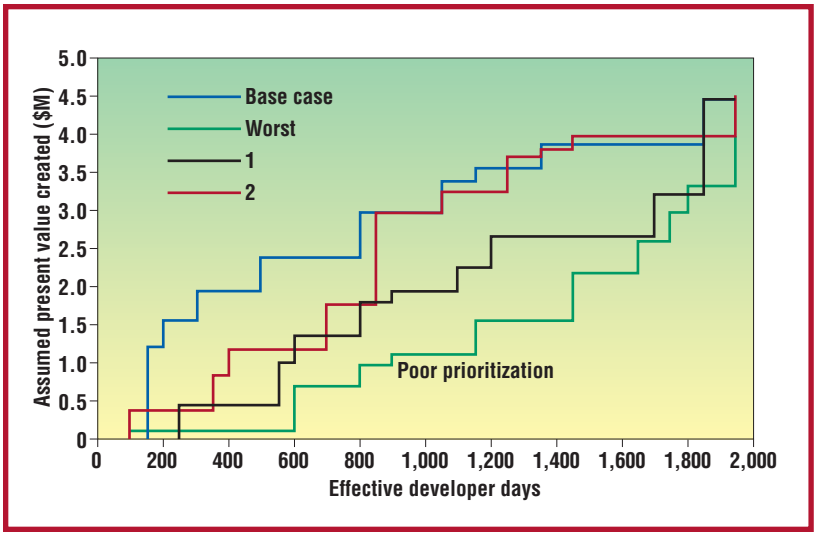


Figure 2. Assumed value creation and prioritization's effect.

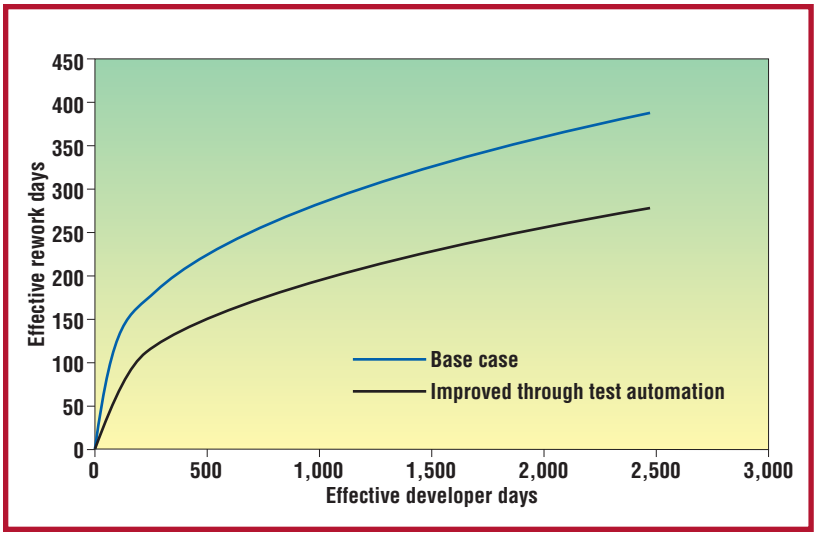


Figure 3. Rework time and process improvement's impact

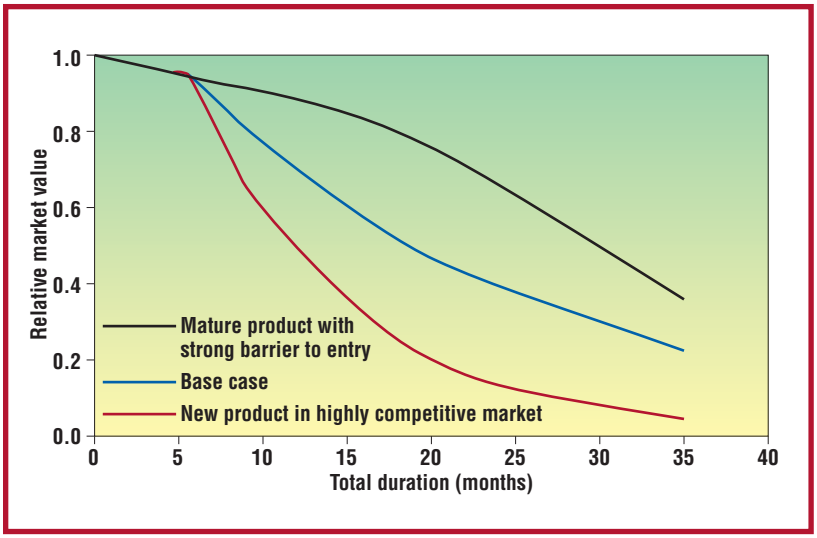


Figure 4. Relative market value capture

Testing and rework required

The time required for application testing depends on both testing new features and regression-testing existing features. We've recorded historical data for many of our projects and found good relationships for mature products, correlating required rework time to effective developer days for that release cycle. If a release's quality target differs from our collected data, then we must modify the curve accordingly. We've also estimated testing rework for each feature and added a base amount for regression. The cumulative rework days can be cross-plotted against the cumulative prioritized development days. Both approaches give similar results. In Figure 3, required regression testing caused the base case's steep slope at the left side of the graph. We've observed that test automation can reduce this overhead for some projects, as the improved curve shows.

Market delay costs

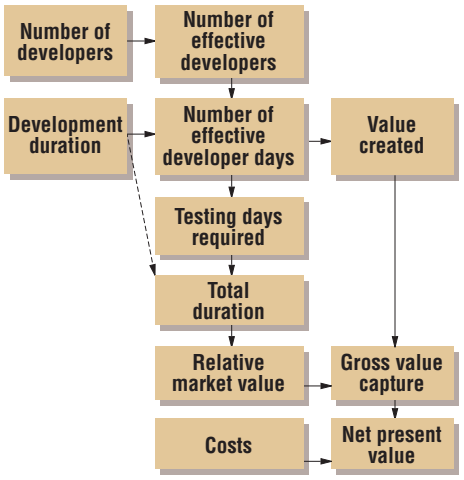
So far, we've assumed that developed material has created value; those who make business decisions are interested in understanding how to translate that into value captured. In Landmark Graphics' case, we're interested in software that's being produced for a general market. We therefore consider *relative market value* to be the fractional value captured as a function of the software's release date. As Figure 4 shows, there's a period of time in which the relative value drops off simply because of money's time value. At some point, competitive threats occur, and market value declines more rapidly. In the case of mature products, this period can be fairly long and the decline relatively flat, as customers don't wish to update their application suite rapidly. For new products, this curve can drop off more precipitously, particularly if substantial competition exists in the new market.

Putting it all together

I can now construct the overall model from these basic components. In addition to the functions I've described, I must define the number of developers on the project and their associated costs. Figure 5 shows a flow diagram of the overall model as a series of table lookups from the parametric curves.

The resulting curves

Figure 6 indicates model results for the base case. The overall *assumed value creation* from



In the equations below, let f_n represent the function underlying the graph in Figure *N*.

- Start with *Number of developers*.
  Number of effective developers = f_1(*Number of developers*)
- Loop over *Development duration*, for example in increments of 20 days.
  Effective developer days = *Number of effective developers* × *Development duration*
  Assumed value created = f_2(*Effective developer days*)
  Effective rework days = f_3(*Effective developer days*)
  Rework duration = *Effective rework days* / *Number of effective developers*
  Total duration = *Development duration* + *Rework duration*
  Relative market value = f_4(*Total duration*)
  Gross value capture = *Relative market value* × *Assumed value created*
  Net present value = *Gross value capture* – *Present value of costs*
- Repeat for more values of *Development duration*.
- Plot *Net present value* as a function of *Total duration*.

Figure 5. Flow diagram of Landmark Graphics' software development process model.

the developed features (red curve) multiplied by the relative-market-value curve gives the gross value captured curve. This represents the amount of created value that can be captured in the marketplace. The shaded area represents the *net present value* after adjusting by cost. The adjusted curve resembles those shown for general new-product development,[6] as both models examine the issues of market timing on value capture. If we treat this as a deterministic problem, it's a simple optimization task to search for the maximum NPV to determine project duration and extract all other project parameters from the model.

This graph also demonstrates the tension that can exist in the various organizational groups focused on assumed value creation or relative market value. The customer, marketing group, or management will be quite cognizant of the relative market value and, as a result, will wish to drive the project up and to the left. The development team, on the other hand, sees that driving the project up and to the right can increase development's assumed value creation. Both parties strive to increase value, but with different motivations. The conflict can be particularly frustrating if either side takes a strong position without understanding the overall picture. This model can help bring both sides to a common understanding by looking to the NPV.

Assessing model sensitivity

One objective was to develop a method for determining the optimal staff size for projects. As Figure 7 demonstrates, we obtain the maximum NPV for the base case project with a

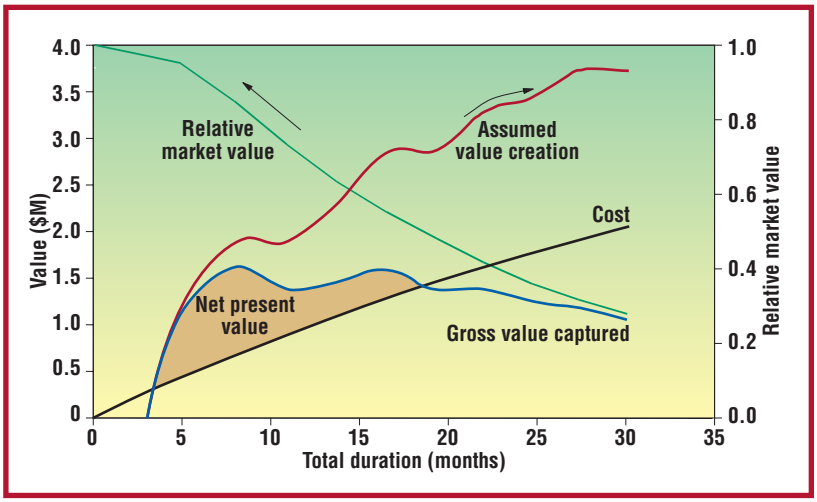


Figure 6. Model results for the base case.

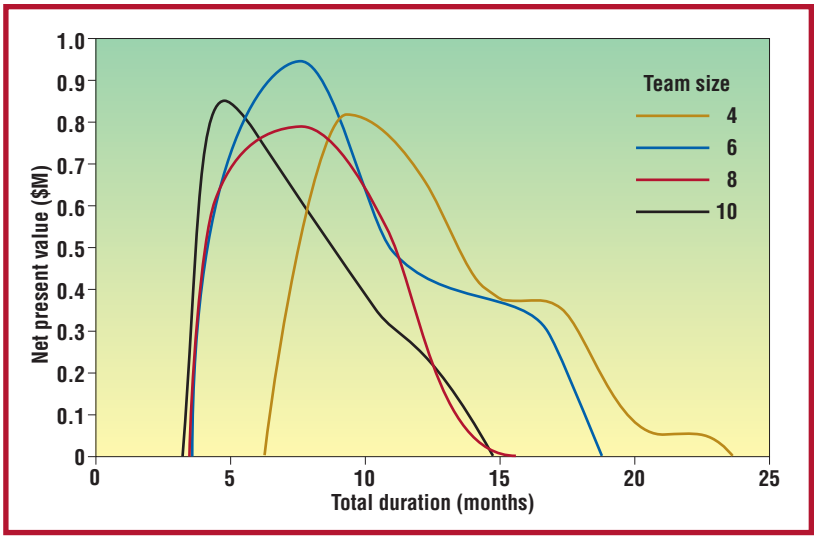


Figure 7. Using the model to optimize team size.

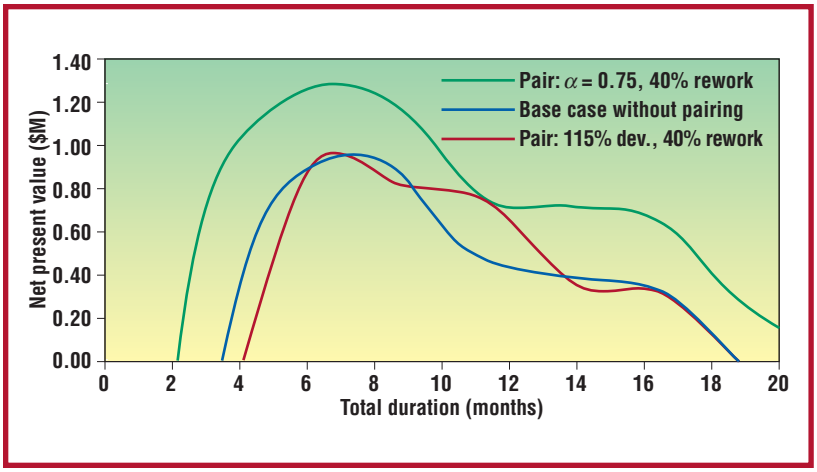


Figure 8. Comparing the base case to a pair-programming case.

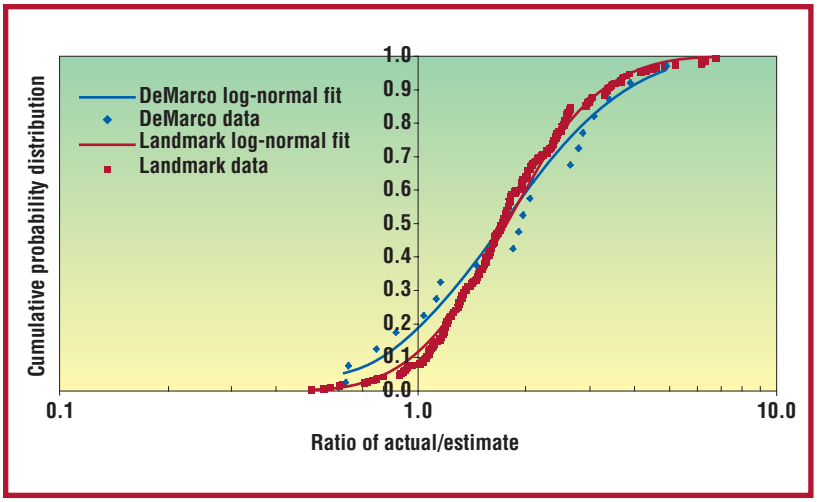


Figure 9. Uncertainty in project estimation.

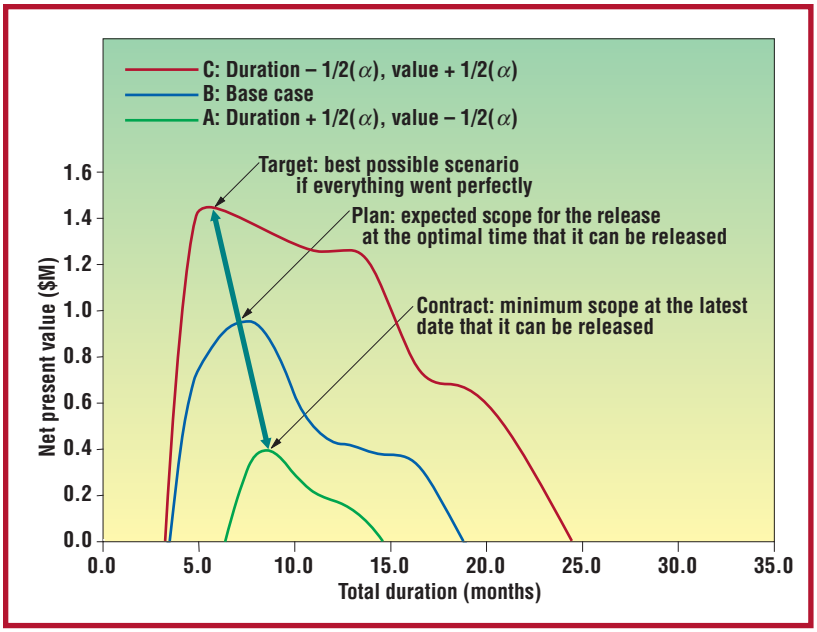


Figure 10. Net present value uncertainty.

team of six. Note that the optimal NPV doesn't vary greatly for different staff sizes, although the corresponding release date and delivered feature sets are different.

We can also change the curves' assumptions and consider some what-if scenarios, such as the economic benefit of introducing pair programming. For our purposes, we want to compare a project with the same team size and same output-quality objectives; this differs slightly from other authors' approaches.[13,14] We compared our base case of six developers to three paired developers.

The model parameters come from Alistair Cockburn and Laurie Williams's research, who reported that paired development takes 15 percent more effort and the defect rate is 60 percent lower.[15] We made the optimistic assumption that fewer defects would cut rework time to 40 percent. As Figure 8 shows, with these assumptions, the base case and red pair curves have nearly identical optimal NPV of about $1 million.

While further reviewing the assumptions, we found that Cockburn and Williams's study compared one-person teams to two-person teams. We looked at team productivity earlier, and, for $\alpha = 0.65$, we have $2^{0.65} = 1.57$, while $2/1.15 = 1.74$. It's possible that pairing's overhead is similar or even less than that observed in general team dynamics. Pair programming advocates claim the overall team is more effective as individuals rotate on different pairing assignments. If this is the case, pairing's impact might be a higher α. The green curve in Figure 8 shows $\alpha = 0.75$ along with the 40 percent factor on rework time. This could be an area of further research.

Uncertainty

Things would be easy if software development were predictable and could be modeled accordingly. However, the reality is that uncertainty is a natural part of the software development process. The Standish CHAOS Report[16] indicates that only 20 percent of projects are finished on time relative to their original plan. We've quantified some of the uncertainty, and in Figure 9 we plot the cumulative-probability distribution of the actual/estimate ratio for our data and compare it to Tom DeMarco's data.[17] We use duration data, while DeMarco reported effort data. The results are remarkably similar and clearly show a log-normal distribution with about a 10 to 20 percent probability of

meeting the original target, about a 50 percent chance of requiring less than two times the original target, and a 90 percent chance of requiring less than four times the original target.

While uncertainty can exist in all the parameters, for most of our projects, we believe the uncertainties in value and duration estimation have the greatest influence on the overall project NPV uncertainty. For a quick view of the uncertainty range, we find it useful to construct three cases for comparison using the standard deviation (σ) for each distribution. Our distribution is log-normal, so $+/-\sigma$ is equivalent to a multiplication scale factor: $+1/2(\sigma)$ is a factor of 1.25, while $-1/2(\sigma)$ is a factor of 1/1.25 = 0.80. Figure 10 shows the NPV as a function of *total duration* for three cases:

A. Duration $+1/2(\sigma)$, value $-1/2(\sigma)$
B. Base case with median values
C. Duration $-1/2(\sigma)$, value $+1/2(\sigma)$

Given that this uncertainty must be managed during the life of the project, we've taken an approach similar to Kent Beck and Dave Cleal's idea of optional scope contracts.[18] We map the three scenarios as

A. Contract
B. Plan
C. Target

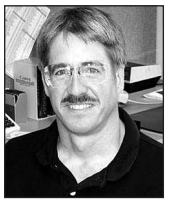
In the Contract, Plan, Target approach, the target, or best possible scenario, gives the most scope at the earliest date possible. This is similar to what Tom DeMarco and Timothy Lister called the "nano-percent date."[19] Teams should have little difficulty establishing targets, since they're based on the assumption that nothing unexpected will happen. The other extreme is the contract. This is defined as the minimum scope at the latest date that can be tolerated. If the project were to have any less scope or be delivered any later, it would be considered a failure. We've experienced a cultural resistance to establishing this project constraint, with project stakeholders unwilling to accept the range of uncertainty. We've used the model's results to help stakeholders understand this range.

By establishing the contract and the project's minimum success criteria, the project team understands what they must absolutely deliver. As long as the stakeholders understand the probability of making the contract, other programs such as a marketing rollout can also be planned accordingly. Just making the relatively quick assessment of these three scenarios provides a wealth of information with which you can make rational investment and planning decisions.

While this model is quite simple, it might be more complex than some organizations require and less complex than others would prefer. To be truly useful, the model should be tuned to the organization and include some subjective assessment. The process of creating the model parameters generates conversations about project assumptions. These conversations can help stakeholders understand the project drivers and success criteria that will help maximize business value.

We've begun running projects using the Contract, Plan, Target approach, and this has significantly reduced our delivery uncertainty. The development team understands what they must absolutely deliver, and the marketing organization can feel confident they'll have a release with content sufficient enough to roll out to customers.

About the Author



Todd Little is a senior development manager for Landmark Graphics. His interests include agile software development, and he's the conference chair for the 2004 Agile Development Conference. He received his MS in petroleum engineering from the University of Houston. He's a member of the Agile Alliance, the IEEE, and the Society of Petroleum Engineers. He is also a registered Professional Engineer in the state of Texas. Contact him at tlittle@lgc.com.

References

1. T. Abdel-Hamid and S. Madnick, *Software Project Dynamics*, Prentice Hall, 1991.
2. B.W. Boehm, *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000.
3. H. Erdogmus, "Valuation of Learning Options in Software Development under Private and Market Risk," *Eng. Economist*, vol. 47, no. 3, 2002, pp. 308–353.
4. H. Erdogmus, "Comparative Evaluation of Software Development Strategies Based on Net Present Value," *Int'l Workshop Economics-Driven Software Eng. Research*, 1999.
5. P.G. Smith and D.G. Reinertsen, *Developing Products in Half the Time: New Rules, New Tools*, 2nd ed., John Wiley & Sons, 1997.
6. F.P. Brooks Jr., *The Mythical Man-Month*, Addison-Wesley, 1975.
7. S.D. Conte, H. Dunsmore, and V.Y. Shen, *Software Engineering Metrics and Models*, Benjamin/Cummings, 1986.
8. C. Jones, *Software Assessments, Benchmarks, and Best Practices*, Addison-Wesley, 2000.
9. M. Denne and J. Cleland-Huang, *Software by Numbers: Low-Risk, High-Return Development*, Prentice Hall, 2003.
10. K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.
11. K. Wiegers, "First Things First: Prioritizing Requirements," *Software Development*, vol. 7, no. 9, Sept. 1999, pp. 48–53.
12. B. Nejmeh and I. Thomas, "Business-Driven Product Planning Using Feature Vectors and Increments," *IEEE Software*, vol. 19, no. 6, 2002, pp. 34–42.
13. H. Erdogmus and L. Williams, "The Economics of Software Development by Pair Programmers," *Eng. Economist*, vol. 48, no. 4, 2003, pp. 283–319.
14. F. Padberg and M. Müller, "Analyzing the Cost and Benefit of Pair Programming," *Proc. 9th Int'l Software Metrics Symp.* (Metrics 03), IEEE CS Press, 2003, pp. 166–178.
15. A. Cockburn and L. Williams, "The Costs and Benefits of Pair Programming," *Extreme Programming Examined*, G. Succi and M. Marchesi, eds., Addison-Wesley, 2001, pp. 223–243.
16. *The Chaos Report*, The Standish Group Int'l, 1998.
17. T. DeMarco, *Controlling Software Projects*, Prentice Hall, 1982.
18. K. Beck and D. Cleal, "Optional Scope Contracts," tech. report, 1999; www.xprogramming.com/ftp/Optional+scope+contracts.pdf.
19. T. DeMarco and T. Lister, *Waltzing with Bears*, Dorset House, 2003.