IT Metrics Strategies

Helping Management Measure Software and Processes and their Business Value

Software Estimation Tricks of the Trade: Secrets They Never Told Me Until I Got to the Real World

Last month's ***ITMS*** discussed the use of metrics in managing deadlines for IT projects. We took two perspectives. One was a forward-estimation view in which we were tasked to forecast the time a project might take, along with effort estimations, staff requirements, and optimization suggestions. The other was a reverse-estimation perspective — starting with a deadline and working the problem backward to quantify what could be done within certain time limits.

The first scenario assumed we had a reasonable handle on the system requirements. The challenge was to establish the makeup of the team and try to achieve the upper echelon of speed and cost performance. This was about aligning the organization's skill resources: applying the right people at the right time to optimize the outcome. Stan Rifkin took on a challenging topic when writing on this issue last month.

In the second scenario, we wrestled with the notion that having the requirements defined is *not* always a given at the onset of a project; however, in most of these cases, we still have a deadline. The challenge becomes how to negotiate the limits of the scope that we can reasonably promise (think triage). We want to make sure that what we commit to is not in the "impossible zone." Looking at industry overrun statistics, it's clear that, as a group, software developers tend to dramatically overpromise.

Continued on page 2.

A Day with Edward Tufte: Visual Displays and the Effective Design of Information

by James T. Heires

It is an ambitious goal to convey the teachings imparted by a veritable master using merely written words. Edward Tufte (pronounced TUFF-tee) is a true showman and a genius when it comes to communication and the display of information. This article summarizes the material presented in Dr. Tufte's fascinating presentation on 25 June 1999 in Minneapolis, Minnesota, USA, so that others may also benefit from his teachings.

Continued on page 7.

executive summary

As we continue to tackle the challenge of managing IT projects with a known deadline and (usually) unknown requirements, the trick is to take what we've learned about documenting our own trends and put it to good use as we face new projects.

We have two sources of information about how software IT projects behave. The first source is our own past performance; the second is observations on projects from metrics research. These tidbits can be combined in ways that can be quite effective for planning. Moreover, they serve to frame project negotiations so that key stakeholders can appreciate the dynamics of such projects, keeping things from spiraling out of control at early, critical stages. My article, "Software Estimation Tricks of the Trade," works at pulling this together.

In the second article, Jim Heires offers an excellent summary of Dr. Edward Tufte's prescriptions for better information design. In the field of IT metrics, a picture says a thousand words. Building the right pictures is the key to meaningful representation of data, allowing effective learning to occur and the right actions to be taken, if necessary.

Finally, Jim Mayes gives us a refreshing profile taken from the front lines of his practice in the Software Engineering Metrics Group. He offers a blueprint on how a group like this targets the achievement of business objectives.

We hope this issue gives you real-world guidance that helps steer you in a proven, reliable direction.

Michael C. Mah, Editor

june 2000 vol. VI, no. 6

CUTTER CONSORTIUM

Continued from page 1.

Getting a stake in the ground about reasonable scope limits within a deadline sparks a conversation that allows you to begin realistic negotiations — negotiations that cater to satisfying what might initially seem to be conflicting interests in multiple dimensions (scope, reliability, and speed). The trick is to understand that the organization has more common interests than it realizes.

That said, I remember something my spouse once cautioned: "Things that seem small can end up big and matter a lot." And, of course, things that seem big can actually turn out to be quite small and matter very little. This started me thinking about metrics-related issues that seem little at first glance, but can have a dramatic impact in the end.

Tidbits from the Stats

Benchmarking statistics collected over the years have yielded some important information about how IT projects behave. If you are aware of these "secrets," they may enable you to better manage projects in the future and avoid known pitfalls. For example, did you know that:

- On average, system testing for an IT applications-development project generally needs to start happening about 70% of the way into a design/code/test phase? So, if you have a project that spans 10 months (for what some call the main build), you should be in systems testing at month 7. This is even more critical if there is significant regression testing with other applications.
- At that point in time, you need to have more than 90% of the code done. By "done," I mean it should be unit coded and integrated/compiled to support system-level testing. If you're not at this 90% level (for example, if you've only built two-thirds of the code to satisfy the requirement), you're probably not going to make the deadline. When faced with this situation, most project teams begin a hasty retreat by cutting scope.
- By that crucial 7th month, you should have found and fixed about 80% of the bugs. You need to use the remaining 3 months to test and debug the defects still left in the code, which should amount to about 20% of the total.
- Going back to the front end of our example project, prior to detailed design and coding (design/code/test phase of 10 months), it's not unusual for the requirements phase to run for about 4 months (about 40%). At that point, detailed design and coding would have started, whereupon the further definition of the requirements might have continued for another 3 months (an additional 30%). It's not unusual for defining the requirements to have spanned 7 months from start to finish (with that last 3 months overlapping with the design/code/test phase). That's 70%.
- For every 100 person-months of effort for design/code/test (proportionally speaking), it is not unusual for 30 person-months of effort to have been expended to define the requirements. During the build phase, it's typical that team size might have been twice that of the staff that was involved during the requirements phase.

Some torpedoes to watch out for:

- If you're pressured to deliver prematurely, say by 2 months (20% of the schedule), you'll release an application that is likely to have three times the number of defects still in the code than if you released it after it was fully tested. If you deploy it 10% sooner (about 1 month in our 10-month example) than it should have been, there are likely to be

Editorial Office: Clocktower Business Park, 75 South Church Street, Suite 600, Pittsfield, MA 01201. Tel: +1 413 499 0988; Fax: +1 413 447 7322; E-mail: michaelm@qsma.com.
Circulation Office: ***IT Metrics Strategies***® is published 12 times a year by Cutter Information Corp., 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA. For information, contact Megan Nields, Tel: +1 781 641 5118 or, within North America, +1 800 964 5118, Fax: +1 781 648 1950 or, within North America, + 1 800 888 1816, E-mail: info@cutter.com, Web site: www.cutter.com/consortium/.
Editor: Michael Mah. Publisher: Karen Fine Coburn. Group Publisher: Anne Farbman, Tel: +1 781 641 5101, E-mail: afarbman@cutter.com. Production Editor: Lori Goldstein, +1 781 641 5112. Subscriptions: $485 per year; $545 outside North America. ©2000 by Cutter Information Corp. ISSN 1080-8647. All rights reserved. Unauthorized reproduction in any form, including photocopying, faxing, and image scanning, is against the law. Reprints, bulk purchases, past issues, and multiple subscription and site license rates are available on request.

about twice the number of bugs still in the code.

- About half of these remaining bugs may result in complete outages and major failures (severity levels 1 and 2).
- If your requirements churned by more than 25%, your schedule might take a 1-month hit (another 10%). It will likely impact the test phase with a surge of defects. The lengthening of the testing will have a bigger impact on effort and cost of the project, because that's when the staffing is usually at its peak in an effort to make the deadline. Effort overruns often are in the range of 20% to 25%. The pain of changing scope can be significant. (This assumes that the scope hasn't grown. If that happens simultaneously, the effects are even worse.)

What Does This Mean?

The knowledge from metrics factoids like these gives us a frame of reference that can help steer discussions about the decisions that need to be made when negotiating up-front terms among managers, IT development staff, end users, and marketing personnel.

What should become apparent from reading this is that while IT projects have complex dimensions, some common truths emerge. Projects with new attributes to meet the competitive needs of the marketplace are research and development in nature. They follow lifecycles that require adequate time for up-front design of a new solution to meet a problem, and they require adequate time for testing. Ironically, these are the two areas that wind up being cut short by "Internet-speed" deadlines and scope growth. Deadline pressure tempts IT staff to rush the code before the design is refined, and testing gets truncated when the deadline hits everyone faster than expected.

We see from metrics data and countless productivity benchmarking engagements that this dynamic is set into motion by overambitious scope — a surefire productivity killer. All indicators demonstrate falling values (along with rising defects) when teams take on too much, get into trouble, and have to beat a hasty retreat at the eleventh hour by pulling out functionality during testing to make the looming deadline.

This was dramatically demonstrated by statistics from the Standish Chaos report, which showed that the 16% of projects surveyed that made cost and schedule targets did so by sacrificing 60% of the original desired scope.

In the face of fixed deadlines, ever-increasing demands for technology, and (therefore) growing scope, we are faced with the negotiation dilemma I addressed in the May issue of ***ITMS*** ("Internet-Speed Deadline Management: Negotiating the Three Headed Dragon"). The question becomes one of how do I find the line in the sand (i.e., how much functionality to promise within a given deadline)?

To answer that question, IT groups can turn to the benchmark trends I described in the March and April issues of ***ITMS***.

Remember Those Benchmark Trends?

Organizations that keep even modest records of projects they completed in the past have a dramatic planning advantage over organizations that don't — even when the projects of the past have different attributes than the one you are trying to estimate.

Because historical profiles show us the limits of achievable speed, they can give us guidance as to what we should realistically promise. If you've never run a two-minute mile, don't promise it — unless you are someone who enjoys reneging on your promises.

Another reason for not overpromising has to do with the quality of your work when it is under the gun. I don't know about you, but I sometimes have to consciously make a choice between doing too many tasks poorly or doing fewer tasks well. Given the choice, I'd like to do the latter.

Beware Excessive Multitasking

From behavioral research cited by people like Rick Jarow (visiting assistant professor of history of religion at Vassar College, former Mellon Fellow in the Humanities, Columbia University), it has become clear that humans cannot reasonably multitask more than four to five things at once. It's been said that the famous science fiction author Isaac Asimov had as many as 10 projects going at once — but I know I can't multitask to this extent, even if I fantasize that I'm Isaac Asimov.

Interestingly, in a number of informal surveys among IT professionals, people report working on as many 11 projects at a time! I've conducted this inquiry of my audiences at every recent speaking engagement. We have taken multitasking to an extreme, hoping that we can parallel process as well as a machine. In the process, we set ourselves up for mediocrity — exactly what perfectionist IT professionals fear the most. Yet, like a moth to a flame, we are drawn to multitask, and our tendencies to overpromise virtually guarantee mediocrity.

SEI Project Estimation Guidelines Contain Good Advice

A number of years ago, the Software Engineering Institute (SEI) released some very helpful guidelines on validating software cost and schedule estimates.[1] Author Robert Park visited numerous companies that had implemented advanced practices for project estimation. To paraphrase, among these processes were the use of demonstrated accomplishments and historical evidence to validate promises on new projects.

Before you start saying things like "That was then, this is now" or "This project is completely different from what we did before," keep in mind that if you don't at least sanity-check your promises, you're at risk of flying right into a brick wall. For example, if on previous six-month projects, you've successfully built systems that accomplished a range of 28-35 work requests by your clients/users, it would be suicide to promise 49. It might even be statistically impossible from your history, especially if the 49 are highly complex. Of course, to not know the previous range of 28-35 means you will not know when to say no as your clients ask for more and more functionality under tighter and tighter deadlines. You will be compressed, I guarantee.

Control your own destiny by knowing the limits of what you've been able to compress before. Keep that tucked away in the back of your mind as you evaluate the potential risks to the project. Better yet, put that range out in the open and on the table — share this information as part of your negotiation strategy. Remember, you must choose between doing fewer things well or multiple things poorly.

Using Benchmark Charts to Bracket the Promised Scope

If you got started late collecting core metrics and benchmarking your organization's past performance, don't worry: it's never too late. Use a short form to scoop even approximate profiles on projects that are relevant to the one now facing you. The data doesn't have to be perfect, as insisting on precision in cases where it's not practical will only hinder you.

Charting even a handful of recent projects will allow you to map the speed with which your teams have pulled off projects. You can then generate the kinds of trends I showed in the May issue or even do it by hand for a quick view. (I've described this technique in previous articles to help organizations triage year-2000 conversion projects, which were basically IT conversion projects with fixed deadlines. The same technique applies here and anywhere that you need to assess what you can build within a given deadline.)

If you generate a quick chart of speed as a function of small, medium, and large projects, it might look like Figure 1. The x-axis should be labeled "size," and is the amount of new and changed functionality. This could be in code, programs, modules, objects, function points, or other units of physical size. Smaller projects on the left, larger projects on the right.

The y-axis is in months for the design, code, build, and test phases. Let's say you build object-oriented applications in C++. If you had a range of sizes from 50 objects (the smallest) to 400 objects (the largest), spanning anywhere from 3 months for the smallest to 11 months for the largest, then you'd chart these coordinates (50, 3) (400, 11).

An upwardly sloping pattern will emerge, most likely showing longer schedules for larger projects. Drawing a line along the lower bound of the sample represents the fastest schedule your projects have exhibited; drawing a line along the upper bound shows the longest schedules your projects have exhibited. Projects with more effective teams, less requirements churn, less turnover,

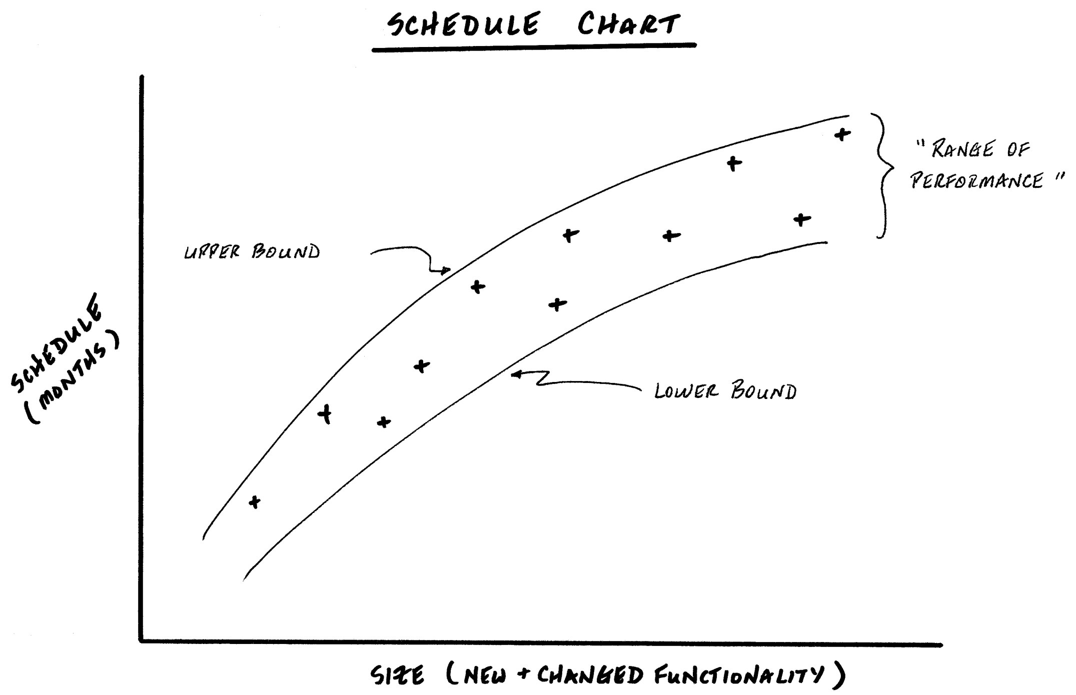

Figure 1 — Chart the schedules you achieve across the size of the projects you build.

and better processes, tools, and methods will tend to be on the lower bound and will have finished faster. All of those projects where Murphy's Law ruled probably hovered around the upper bound and took longer. If you consider these factors when you carefully assess the things in your favor and the possible traps on your upcoming project, I congratulate you — you've just started practicing risk analysis.

Now that you have a chart that sketches out the upper and lower ranges, you can place a ruler at any given deadline — for example, eight months. You can then pick off the effective high and low of what you've been able to do before. Easy, right? This visually demonstrates a range of values, allowing you to see the best-case/worst-case outcomes.

For an eight-month deadline, you might be able to say, "In the past, in the *best* of circumstances, we've been able to build X number of objects to accommodate 35 work requests. In the *worst* case, we've been able to build Y number of objects to accommodate an equivalent of 28 work requests." You can then negotiate within the range of 28 to 35.

Do the Same with Effort Charts

The next step is to chart the same type of graph for effort, using person-months on past projects. For example, the project that delivered 50 objects was built by four people (full-time) over the 3-month schedule. That means that 12 person-months of effort were expended. The coordinate is (50, 12).

The project of 400 objects over 11 months had 10 people from start to end, working full-time. 110 person-months were expended (11 times 10). That coordinate is (400, 110).

Another trend will emerge, one that might look like Figure 2. Notice that both trends are nonlinear. That is to say, projects that are 50% larger than the previous do not necessarily take 50% more time and 50% more effort. This is a virtual law of nature on IT projects, and it is vital to be aware of so that you don't engage in simple extrapolations that would get you in trouble.

What this does is give you another sanity check. If you have 8 people assigned to a project with a deadline of 8 months, you can use your ruler to determine the range of project scope that 64 person-months of effort have been able to pull off in the past. Having another dimension of information gives you yet another angle on how much to

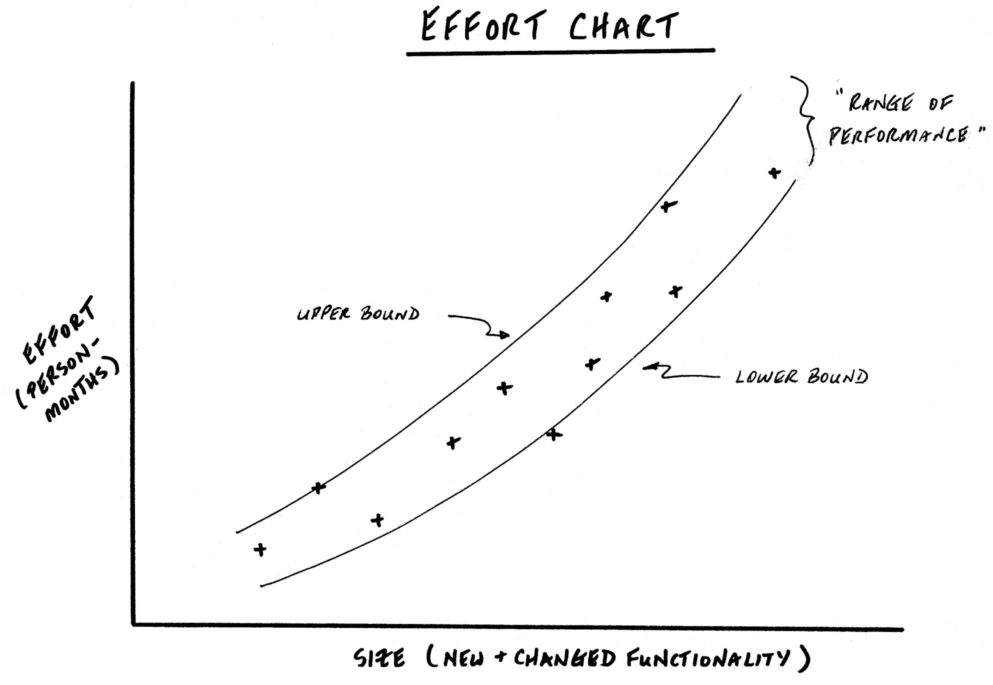

Figure 2 — Chart the effort spent across the sizes of projects you build.

realistically promise. Given IT staff shortages and the urgency of projects today, we tend to budget too little effort to build too much functionality. Understaffed teams wind up being pressured to work more overtime, and burnout becomes a large factor. Effort overrun statistics (from QSM Associates' figure of 127%, to as much as the 189% figure reported by Standish) show that we often set ourselves up with unrealistic effort expectations that are not achievable.

Tie It All Together and Say It with Conviction

You have just successfully sanity-checked any promise that you might decide to make. Combined with the estimation tidbits and rules of thumb I described earlier, you can construct a more informed negotiating position. Knowledge is indeed power.

If you've been feeling disempowered and delegitimized during negotiations in the past, this will help you build a more effective case. Having data can give you what you need to persuade others in the organization to agree on joint commitments to satisfy as many common interests and goals as possible. Nobody wins if decisions are made that set the stage for the chaos and overruns our industry has become known for. Dysfunctional dynamics hurt both developers and the end users we so desperately want to please (consciously or unconsciously).

If you're feeling lucky, perhaps you'll promise more equivalent work requests than you've done before, but be careful. If you choose this, do it from an informed perspective. In the absence of knowing the range of what you've pulled off before, you have no metrics gauge of what to promise, outside of pure instinct.

To avoid this, get some numbers to help serve as a guide. We do it everyday, with stock market indexes to guide our investments, medical numbers to help guide our diet and exercise, and dashboard gauges to drive our cars. By using simple benchmark trends that you can build without a consultant, and combining these with demonstrated patterns of IT projects from industry statistics, you can take numbers and translate them into pictures to make better, more informed decisions. Taking various perspectives with regard to deadlines, effort, defects, and promised functionality gives us an appreciation for the multidimensionality of IT projects.

The truth is, we want to do better and not overpromise. But remember that, if you're like most IT professionals, you probably tend to do just that. Ultimately, we are the ones that decide, either actively, or passively, to either do fewer things well or many things poorly. Which will you choose?

References

[1]"Checklists and Criteria for Evaluating the Cost and Schedule Estimating Capabilities of Software Organizations," Special Report CMU/SEI-95-SR-005, January 1995. "A Manager's Checklist for Validating Software Cost and Schedule Estimates," Special Report CMU/SEI-95-SR-004, January 1995.

Resources

Karten, Naomi. *Managing Expectations: Working with People Who Want More, Better, Faster, Sooner, Now!* Dorset House, 1994.

McConnell, Steve. *Rapid Development*. Microsoft Press, 1996.

DeMarco, Tom. *Controlling Software Projects*. Prentice Hall, 1982.

Putnam, Lawrence, and Ware Myers. *Industrial Strength Software*. IEEE Computer Society, 1997.

A Day with Edward Tufte: Visual Displays and the Effective Design of Information

Continued from page 1.

The Two Problems of Information Display

There are two problems that arise when attempting to display information. The first is the difficulty in representing the multivariate nature of much of the information we wish to display. Multivariate is the term used by statisticians to describe relationships with three or more dimensions. For example, the four dimensions of latitude, longitude, altitude, and time can describe an airplane's movement through space. The problem in displaying multivariate information is that most of our modes of display are flat.

In Euclid's 1570 work, *The Elements of Geometrie*, very effective "pop-up" models of various geometric shapes (pyramids, etc.) accompany descriptions of solid geometry. Tufte displayed a copy of Euclid's masterwork to illustrate its novelty. Euclid succeeded in escaping the flatland of the printed page more than 400 years ago! Tufte does the same today in his work. In fact, Tufte has reproduced Euclid's 3-D paper constructions of solid geometry in his own work.[1]

The second problem of information display is that high-resolution information cannot be effectively displayed using a low-resolution medium (e.g., computer screens or overhead projectors). Resolution is a measure of data density and can be expressed in terms of pixels per square inch (for example).

Information resolution problems surface frequently when dealing with continuous data. Nuances in color, temperature, or time tend to vanish with the use of a low-resolution medium.

Tufte offers five principles to help resolve the two problems of information display.

The Five Grand Principles of Information Display

1. Enforce visual comparisons. This principle helps to visually address the question, "Compared with what?" Normally, quantitative information displays an attempt to answer the question "How much?" "How fast?" or "How good?" This additional dimension conveys a sense of scale to the analysis. One way to create visual comparisons is to apply spatial orientation within the reader's field of view. Forming an adjacent juxtaposition, as opposed to stacking images in time (aka one damn thing after another), is one way to carry out this principle. The single most powerful technique is that of the small multiple. The small multiple is a series of small images that are intended to answer questions directly by visually enforcing comparisons of change. This technique communicates multivariate information within one eyespan. A simple example of this is Eadweard Muybridge's small multiple depicting the childhood game of leapfrog.[2] Here, within one eyespan, one can fully understand how to play leapfrog.

The multiple variables of the relative position of two individuals and how they change over time are fully described in one small multiple (with no words). This technique has the added benefit of lending integrity to the presentation.

2. Show causality. Tufte advocates showing the relationships that exist between data. One example can clearly illustrate the importance of showing causality; Charles Joseph Minard's image of Napoleon's march to Moscow during the war of 1812 shows the effect of temperature on the death toll of Napoleon's army.[3] One particularly telling detail of this figure can be seen at the intersection of the army's retreat and the (then frozen) Berezina river, where 22,000 soldiers fell through the ice and drowned. This display clearly shows the link between temperature and the number of surviving soldiers!

3. Capture multivariate information. Rich, complex, information-packed graphical displays can be the most elegant forms of communication. The "cyclogram" of Russian Cosmonaut Georgi Grechko illustrates this principle quite well.[4] Grechko and his comrade Yuri Romanenko orbited the earth in Salyut 6 for 96 days (December 1977 to March 1978). During the mission, a meter-long chart was created to record various events of the flight. Information such as elapsed time, transitions between day and night, orbit duration, and planned and actual daily activities are among the notations on the cyclogram. There were 22 variables recorded on this one multivariate chart!

4. Integrate text, image, number, and figure. To form a coherent argument, combine all these elements into one design. Almost 400 years ago, Galileo Galilei effectively used text/image integration to document the first telescopic images of Saturn, by simply hand-drawing images of Saturn amongst his Italian notation.[5]

If it weren't for John Snow's careful investigation and two-dimensional spatial comparison, the death rate due to cholera (250 years later) would certainly have been much higher.[6] Snow combined the number of deaths due to cholera, the location of those deaths, and the location of a suspect well. It was this clever display of information that formed the basis of Snow's argument. The eventual removal of the Broad Street pump handle served as the decisive event in the cure to the epidemic.

5. If numbers are boring, design won't help. The quality, integrity, and relevance of content are far more important than clever design.

The purpose of any display of information is to reason about the content, not to marvel at the design!

Use the Smallest Effective Difference

Design elements such as color, texture, and perspective should be used only to enhance reasoning about the content. There is a tendency to overuse design elements simply because they are readily available. Computers afford us myriad colors, pointers, backgrounds, borders, and grids with which to express ourselves. The design strategy of the smallest effective difference urges us to make subtle visual distinctions, while maintaining clarity. Cartographers have been practicing this design strategy for centuries; for example, using shades of color in topographical maps to illustrate altitude.[7]

Overuse of design elements leads to clutter (what Tufte calls "chart junk") that tends to confuse the message.[8] The tabloid press is famous for using chart junk such as stacking coffins in a bar chart format to illustrate the death rate of the London cholera epidemic.[9]

Clutter is not the fault of content — rather, it is the fault of design.

Financial Displays

Financial displays attempt to graphically depict economic changes over time. Recurring problems with this type of presentation can also be solved with simplifying techniques. In general, showing the zero point adds little benefit to a display of nonzero values. Show more horizontal, rather than vertical, data to effect understanding.

Reveal variation along with the mean to prevent a phenomenon called regression toward the mean. Extreme random events (so-called special causes) tend to normalize toward the average value. Showing the variation brings the proper context to the display.

Statisticians routinely use the X-bar and R chart to defeat regression toward the mean.

Two dubious qualities of financial data display recur from time to time: that which is unadjusted for inflation and that which is unremarkable. Tufte calls these qualities "the lie and the bore" and urges their ouster. All financial data displayed over any length of time should be adjusted for inflation to avoid "the lie" of ignoring the effect of inflation. The unremarkable seasonal fluctuations in many presentations are mostly of little interest. Seasonally adjusted data removes "the bore" from the regular (and thus, unremarkable) variation that always occurs (the retail sales in December, for example).

A 12-Step Program for Running Meetings

When it comes to holding meetings or giving presentations, there are several ways to improve one's effectiveness:

1. Show up early. This one tip can frequently prevent certain disaster — or at least give you that last bit of ever-important preparation time. Use this time wisely, ensuring that the meeting room is prepared, the equipment is in place (and working), and you are familiar with the environment. If anything should need attention, there is still time to make corrections.

2. Start off right. Never apologize! One blunder that can ruin your credibility right from the beginning is to say something like "Sorry I'm late" or "I always get nervous in front of crowds." Also, avoid first-person singular form — it sounds amateurish. Remember three things to avoid these problems: practice, practice, practice!

3. When explaining a complex figure, follow the particular-general-particular principle.[10] Give a brief introduction of the figure and show a particular detail. Describe the detail, then step back and explain the general architecture of the figure. Finally, wrap up with a discussion of another particular detail. This approach explains the general organization of the figure and reinforces your message with two specific examples.

4. Give everyone at least one piece of paper. Paper is the highest resolution medium; use it to your advantage to make your point. In addition, paper leaves traces of who said what, when, and where.

5. Avoid overheads with bulleted lists. Overhead projectors are one of the lowest resolution forms of communication. The all-too-typical lists of bulleted text do very little to clearly and completely tell your story.

6. Don't patronize your audience. Treat the audience as colleagues who are interested in helping you solve a problem (they just might be). Strive to be simple, but not simple-minded. Your presentation should allow the audience to reason about the content — make sure you have some.

7. Use humor only to build your point. Effective use of humor is difficult to master in business settings. If you are not comfortable with either your audience or the use of humor, leave it out. If the use of humor supports your point without offending members of your audience, use it judiciously.

8. Avoid the use of gender-centric pronouns. Keep your message gender neutral to appeal to a wider audience and avoid political problems.

9. Wait for questions. When the time comes to entertain questions, ask for them and then wait 10 seconds for responses. This may feel like an eternity while you are in the spotlight, but if you don't wait long enough, some audience members will not ask their burning questions and might leave the meeting disappointed in your presentation.

10. Show your enthusiasm. If you believe in the stuff, let your audience know it! Don't hide behind a lectern with your hands at your sides. Use gestures, voice inflection, and other nonverbal communication to share your enthusiasm with your audience.

11. Avoid dehydration. Drink water before and during your presentation, especially if you travel by airplane to get to your meeting. In addition to dry mouth, dehydration can cause rapid heart rate, low blood pressure, and lightheadedness; you don't need these symptoms on top of the normal "butterflies in the stomach." Water contains essential electrolytes, which cause energy conversion to take place at a cellular level.

12. Finish early. Have you ever heard someone say, "It would have been a better meeting if it had just lasted another 30 minutes?" Everyone will be happier if you end earlier than announced. Ending late gives the message that you were ill prepared and out of control.

References

[1]*Envisioning Information*. Edward Tufte. Graphics Press, 1990, p. 16.

[2]*Visual Explanations: Images and Quantities, Evidence and Narrative*. Edward Tufte. Graphics Press, 1996, pp. 108-109.

[3]*The Visual Display of Quantitative Information*. Edward Tufte. Graphics Press, 1992, p. 176.

[4]*Visual Explanations*, pp. 92-93.

[5]*Envisioning Information*, p. 120.

[6]*Visual Explanations*, pp. 27-37.

[7]*Visual Explanations*, pp. 76-77.

[8]*The Visual Display of Quantitative Information*, pp. 107-121.

[9]*Visual Explanations*, p. 37.

[10]"Classroom and Platform Performance." *The American Statistician*. February 1980, p. 13.

About the Author

James T. Heires is a 13-year veteran of the software industry; the majority of his time was spent with Rockwell Collins, Inc. His professional experiences include design of electronic flight instrumentation, flight management systems, and consumer electronics. Mr. Heires's work in software process improvement culminated with the achievement of Software Engineering Institute's Capability Maturity Model Level 3 in two Rockwell Collins business units. Currently, Mr. Heires is improving the state-of-the-practice in project cost estimating using parametric modeling techniques. Mr. Heires received recognition in 1998 from *WHO'S WHO of Information Technology*, and writes software product reviews for *Application Development Trends* magazine. He can be reached via e-mail at jtheires@collins.rockwell.com.

Achieving Business Objectives II: Building a Software Metrics Support Structure

by Jim Mayes

"There are three kinds of people: those who can count and those who can't." — Anonymous

Whether you are trying to initiate a software metrics program or trying to sustain one due to organizational changes, budget constraints, or other factors, the benefits must be continuously *sold* within the IT organization. Software metrics is often perceived as intrusive, therefore, the benefits must be understood in relation to achieving business objectives, because that is the bottom line.

In my last article (in the March issue of ***ITMS***), I addressed the software estimation relationship between achieving business objectives and balancing time, cost, and quality (TCQ) with business value.[1] This article provides a high-level blueprint for building a metrics structural model needed to support that concept. This includes establishing a Software Engineering Metrics Group (SEMG). Just as the Carnegie Mellon University/Software Engineering Institute's (CMU/SEI) Capability Maturity Model (CMM) establishes a Software Engineering Process Group (SEPG) for managing software processes, an SEMG should be established for managing software metrics. The SEPG and SEMG functions complement each other and have common goals in relation to achieving business objectives: continuous improvement, risk management, and balancing TCQ.

Background

In 1994, after 20 years as a software programmer and development manager, I read Ed Yourdon's book, *The Decline and Fall of the American Programmer*. It was an epiphany for me, as if someone had opened the curtains and turned on the lights. The opportunities seemed endless, and still do, because *there is no end to continuous improvement.* Since that time, I have served as a member of an SEPG, responsible for software project planning processes, primarily estimation, and as an internal estimation/metrics consultant.

Why did I focus on estimation? As a software development manager for many years, I found that one of the hardest and most critical tasks that I had to undertake was providing credible software project estimates. Our basis for estimation had been religious in nature; i.e., faith in our ability to deliver, rather than based on hard data. As one of my metrics consultant coworkers once said, "Before our metrics program began, we provided *hysterical* estimates instead of *historical* estimates." Therefore, as I focused my energy on process improvement related to software estimation, I realized that software metrics is an inseparable part of estimation, continuous (process) improvement, and risk management. As Lawrence Putnam and Ware Myers said, "An organization can accomplish its estimate only if it has a software development process it can repeat."[2]

So You Can Measure? So What!

At every International Function Point Users Group (IFPUG) conference, Bill Hufschmidt (Development Support Center, Inc.) hands out buttons with witty phrases on them. At the fall 1999 conference, he gave out buttons that said "So You Can Measure? So What!" I loved that button. This is what it means to me:

> Merely measuring things or counting function points isn't what software metrics is all about. It is what you can do with the information, and the metrics derived from the measures, that matter. When metrics are only used for senior management reporting at the macro level, the primary benefit is lost. To maximize the potential benefit, metrics must provide meaning at the *project level*. Within the global scheme of software metrics, the rubber hits the road at the project level. All other metrics should be traceable to project metrics so they can be put to work and can add value to the overall software development process.

In this context, it is important to understand the definition of what we mean by measure, metric, and indicator (MMI). Basically, a measure is a standard unit of measurement, such as function points for size, or hours for effort. A metric combines two or more measures into a meaningful derivation, such as function points per hour for productivity. An indicator is something that draws attention to a particular situation, such as a flag or deviation outside predetermined tolerances or control limits. These MMI terms distinguish the difference between software metrics and project management status reporting.[3]

It is also important to understand these MMI terms in relation to how software metrics is perceived by software organizations being measured. In his book *Why Does Software Cost So Much?* Tom DeMarco wrote, "I observe there are at least three different reasons we collect metrics, as follows:

- To discover facts about our world
- To steer our actions
- To modify human behavior"[4]

DeMarco goes on to suggest a metrics spectrum, as shown in Figure 3, with behavior modification on the left, steering in the middle, and discovery to the right. The objective of the SEMG is to provide discovery metrics for continuous improvement and steering metrics for risk management. DeMarco sug-

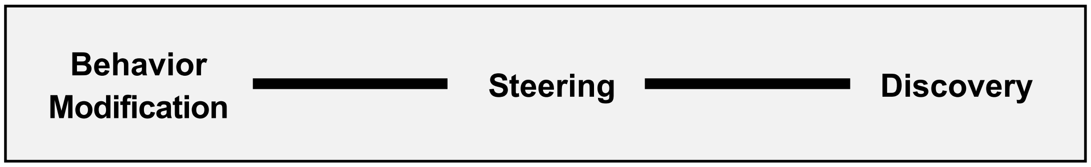

Figure 3 — DeMarco's metrics spectrum.

gests that the further to the right you are on the metrics spectrum, the more successful your use of metrics will be. As you move toward the left side of the spectrum, you must beware of the Hawthorne effect, where people respond differently just because they are being measured. An example of behavior modification that you don't want was illustrated in a *Dilbert* comic strip several years ago. In the first frame, the pointy-haired boss said that he was going to provide a "reward of one dollar for every defect" each programmer discovered. The second frame showed Wally all excited about "coding a new minivan."[5] As this illustrates, a software metrics program should focus on the connection between steering and discovery, staying away from behavior modification metrics of this type. It should be clear that the MMIs are related to continuous improvement, managing risk, and balancing TCQ.

Blueprint for Building a Metrics Support Structure

A multilayered support structure for achieving business objectives is shown in Figure 4, with each of the four layers building on the layer below it. Each layer is also dependent on the other layers for achieving business objectives, just as a building is dependent on its foundation, floor, walls, and roof for it to be structurally sound and to achieve its functional goal. I believe there are many similarities in managing the construction of buildings and software. (Okay, so I have a degree in building construction from an engineering school, and *This Old House* is my favorite television program.) How easy do you think it would be to construct a building without any measures? It would be impossible, right? Of course, software products can be constructed without measures or processes. There are also plenty of industry figures showing the high percentage of project failures each year. I'm sure there is an equally high percentage of projects that are completed but fail to provide the appropriate business value because of cost or schedule overruns. There are also plenty of projects that never get initiated, due to bad estimates that falsely indicate that the project will cost too much or take too long. It is important to quantify project results for empirical analysis in order to manage these risks, improve estimates, improve processes, and assess business value attainment.

The environment needed to support this initiative consists of the following structural components (layers):

- Continuous improvement — the foundation
- Risk management — the floor
- Project management — the walls
- Balancing TCQ for achieving business objectives — the roof

The SEMG provides support related to each of these components, as described in the paragraphs that follow.

Foundation: Continuous Improvement

An SEMG that is established in relation to a continuous improvement program and a process management group (SEPG) lays the foundation for achieving business objectives. The concept of establishing a group for managing CMM measurement practices is supported by CMU/SEI-92-TR-25 technical report, "Software Measures and the Capability Maturity Model," by John H. Baumert and Mark S. McWhinney, 1992, which states:

> Many of the potential benefits that an organization can derive from a sound measurement program are often not achieved due to a half-hearted commitment to a measurement program. The commitment cannot be just a policy statement; it must be total commitment. The policy must be followed with the allocation of resources to a measurement program. This includes allocating staff as well as tools. It is important that an individual or group be assigned responsibility for the measurement program. This group identifies the measures to be collected, establishes training in measurement, monitors the consistency of the measures across the projects throughout the organization, and can help initiate measures. This group can also provide composites of historical data that managers can use in planning and monitoring projects. The human element in a measurement program should not be taken lightly. Success and failure are tied to people.

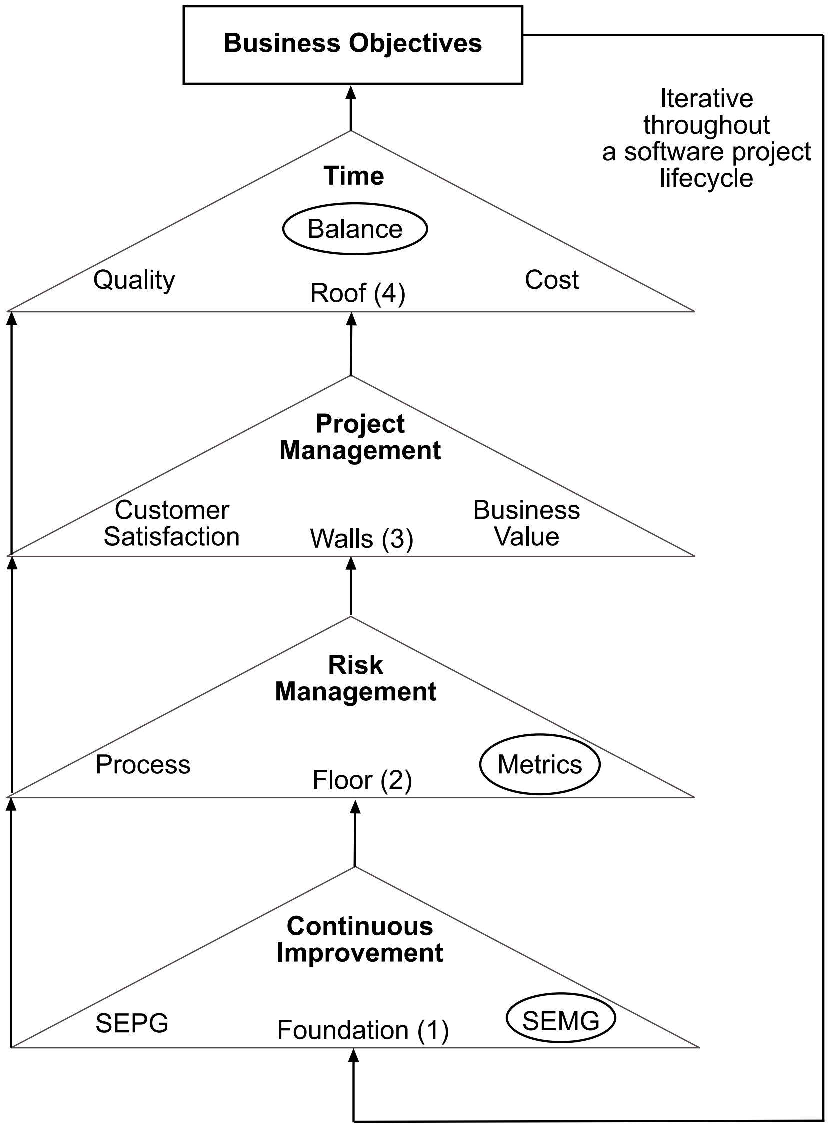

Figure 4 — Software metrics support structure for achieving business objectives.

> All staff members need to see the benefits of the measurement program and understand the results will not be used against them.[6]

Another technical report, CMU/SEI-97-HB-003, "Practical Measurement for Process Management and Improvement," by William A. Florac, Robert E. Park, and Anita D. Carleton, "shows how well-established principles and methods for evaluating and controlling process performance can be applied to a software setting to achieve an organization's business and technical goals." The report focuses on process management related to operational software processes. Its primary focus is on the "enduring issues that enable organizations to improve not just today's performance, but long-term success and profitability of their business and technical endeavors."[7]

These CMU/SEI reports describe different aspects of software measurement and metrics that should be managed by the SEMG for achieving business objectives. As stated in CMU/SEI-97-HB-003, "Every organization asks the question, 'Are we achieving the results we desire?' This gets couched in many ways, such as: Are our customers satisfied with our product and services? Are we earning a fair return on our investment? Can we reduce the cost of producing the product or service? How can we improve the response to our customers' needs or increase the functionality of our products? How can

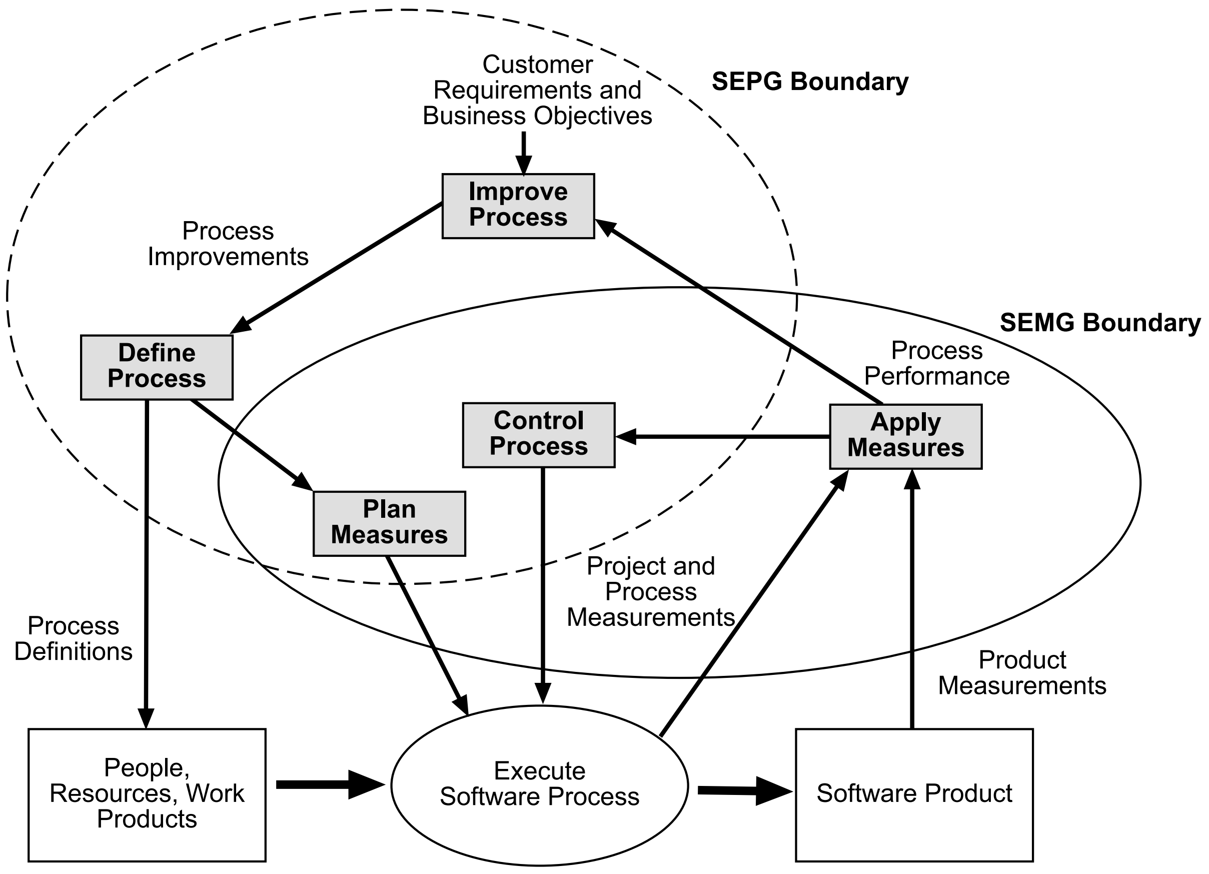

Figure 5 — Process management relationship between continuous improvement, the SEPG, and the SEMG.

we improve our competitive position? Are we achieving the growth required for survival?" The SEMG role in this layer is shown in Figure 5, which illustrates the process management relationship between continuous improvement, the SEPG, and the SEMG, for answering the questions. It is a chart provided in CMU/SEI-97-HB-003,[7] which I have adapted to show the SEPG functional boundary, the SEMG functional boundary, and where the boundaries overlap.

The SEMG provides the measurement capability and expertise necessary to analyze process improvement opportunities using statistical analysis techniques for process control, root cause analysis, and solution analysis, which include:

- Control charts
- Scatter diagrams
- Run charts
- Cause-and-effect diagrams
- Histograms
- Bar charts
- Pareto charts

Floor: Risk Management

"Risk management is project management for adults."[8] This was the title of a keynote presentation that DeMarco gave at the 1995 Quantitative Software Management, Inc. users conference. In the presentation, he stated that the software industry today is developing process sophistication and orderliness via the CMM; however, the use of the term *maturity* in the CMM is deviant to the term that he would use to recognize project management: *adulthood.* A great manager is one that deals with uncertainties, not certainties, and this is the fundamental reason that we are not mature. To effectively manage a software project, managers must be aware of the real world and must have the capacity to deal with the real world. The SEMG can help quantify and identify these uncertainties, via discovery and steering metrics, so they can be managed.

If you go back to Figure 4 on page 13, you see that the risk management layer, which includes process and metrics, adds a floor onto the foundation layer, supported by the functions of the SEPG, SEMG, and continuous improvement. Of course, one of the primary reasons for implementing CMM processes is to reduce risks associated with software projects. Therefore, the SEPG's primary role in this layer is to reduce risk by ensuring adherence to process, which includes a risk management process as well. The SEMG role is to provide statistically quantifiable data for managing risks related to project estimates and indicators for triggering alarms during a software project's lifecycle. DeMarco describes the circular definition of risk as follows:

> A risk is a problem that has not occurred. A problem is a risk that has occurred. A transition is a problem going from risk to problem; i.e., the Trucker's Maxim: behind every bouncing ball comes a running child (transition), apply brakes when you see the ball, and do not wait until you see the child.[8]

The triggers and alarms (indicators) provided by the SEMG can identify transition and help prevent a risk from becoming a problem. Project managers will then be able to "apply brakes when they see the ball" (indicator). The SEMG functions that facilitate this ability include:

- Validation of top-down versus bottom-up estimates for identifying risks related to estimates
- Project monitoring of estimated versus actual data (size, schedule, effort, defect rates, etc.)
- Predictive analysis or statistical inference
- Risk analysis related to project tracking

Walls: Project Management

The triangular relationship in this layer includes project management for providing customer satisfaction and business value associated with the business objectives of the software project. The processes and metrics supported in the continuous improvement layer (the foundation) and risk management layer (the floor) carry forward to the project management layer of the structure. With the foundation and floor to build upon, the project management layer builds the walls (or software product) that will support the roof (TCQ) of the structural model. This layer operationalizes software processes and metrics and includes the following steps in the TCQ process[1] that are supported by the SEMG and software metrics:

- Identification of business objectives and project constraints
- Project data collection and analysis
- Top-down estimation for balancing TCQ
- Phased estimation and project monitoring
- Partnering negotiation

Roof: Balancing TCQ for Achieving Business Objectives

Just as a roof completes the structure of a building, this layer is the culmination of the structural components below it; this illustrates the goal of balancing TCQ in relation to business objectives. As stated in my article in the March ***ITMS***: "The objective of any software project is to optimize time, cost, and quality, relative to the expected business value to be received from the software product." The layers below this level — continuous improvement, risk management, and project management — are essential for successfully accomplishing this goal. The SEMG quantifies and analyzes the overall project results as they relate to achieving business objectives. This analysis leads to the identification of continuous improvement opportunities related to software processes and predictive analysis for risk management.

Conclusion

A software metrics program must be sold within the IT organization to get it started and to keep it going. The benefits must be justified in relation to business value and business objectives. The structural model that was presented in this article was intended to illustrate how software metrics relates to business objectives. Establishing an SEMG in this context provides the focus, continuity, experience, backup, and reinforcement of professional skills needed for sustaining and improving an organization's management capabilities for achieving business objectives. To provide these benefits, it must be a centralized function staffed by dedicated metrics and estimation specialists.

Without a metrics program, it would be difficult, if not impossible, to evaluate whether business objectives are being met or to determine the benefit of process changes. I'll close with a quote from DeMarco, which I think sums this up from a business management perspective: "You can't control what you can't measure."[9]

References

[1]Mayes, Jim. "Achieving Business Objectives: Balancing Time, Cost, and Quality." *IT Metrics Strategies*, Cutter Information Corp., March 2000.

[2]Putnam, Lawrence H., and Ware Myers. *Controlling Software Development*. IEEE Computer Society Press, 1996.

[3]Ragland, Bryce. "Measure, Metric, or Indicator: What's the Difference?" *CrossTalk* (www.stsc.hill.af.mil), March 1995.

[4]DeMarco, Tom. *Why Does Software Cost So Much?* Dorset House Publishing, 1995.

[5]Adams, Scott. *Dilbert*. 1996.

[6]Baumert, John H., and Mark S. McWhinney. *Software Measures and the Capability Maturity Model*, CMU/SEI-92-TR-25, ESD-TR-92-24, September 1992.

[7]Florac, William A., Robert E. Park, and Anita D. Carleton. *Practical Software Measurement: Measuring for Process Management and Improvement.* CMU/SEI-97-HB-003, 1997.

[8]DeMarco, Tom. *Risk Management: Project Management for Adults*. Presentation, QSM Users Conference, May 1997.

[9]DeMarco, Tom. *Controlling Software Projects*. Yourdon Press, 1982.

About the Author

Jim Mayes is currently an estimation consultant with BellSouth and is a certified function point specialist. He has more than 26 years of experience in software development and lifecycle management. Over the past five years, he has been directly involved in providing quantitative software project estimation, data analysis, and metrics related to software process improvement and outsourcing initiatives at BellSouth. Mayes can be reached at www.softwareestimator.com; Tel: +1 770 587 4219; E-mail: Jim.Mayes@bridge.bellsouth.com.