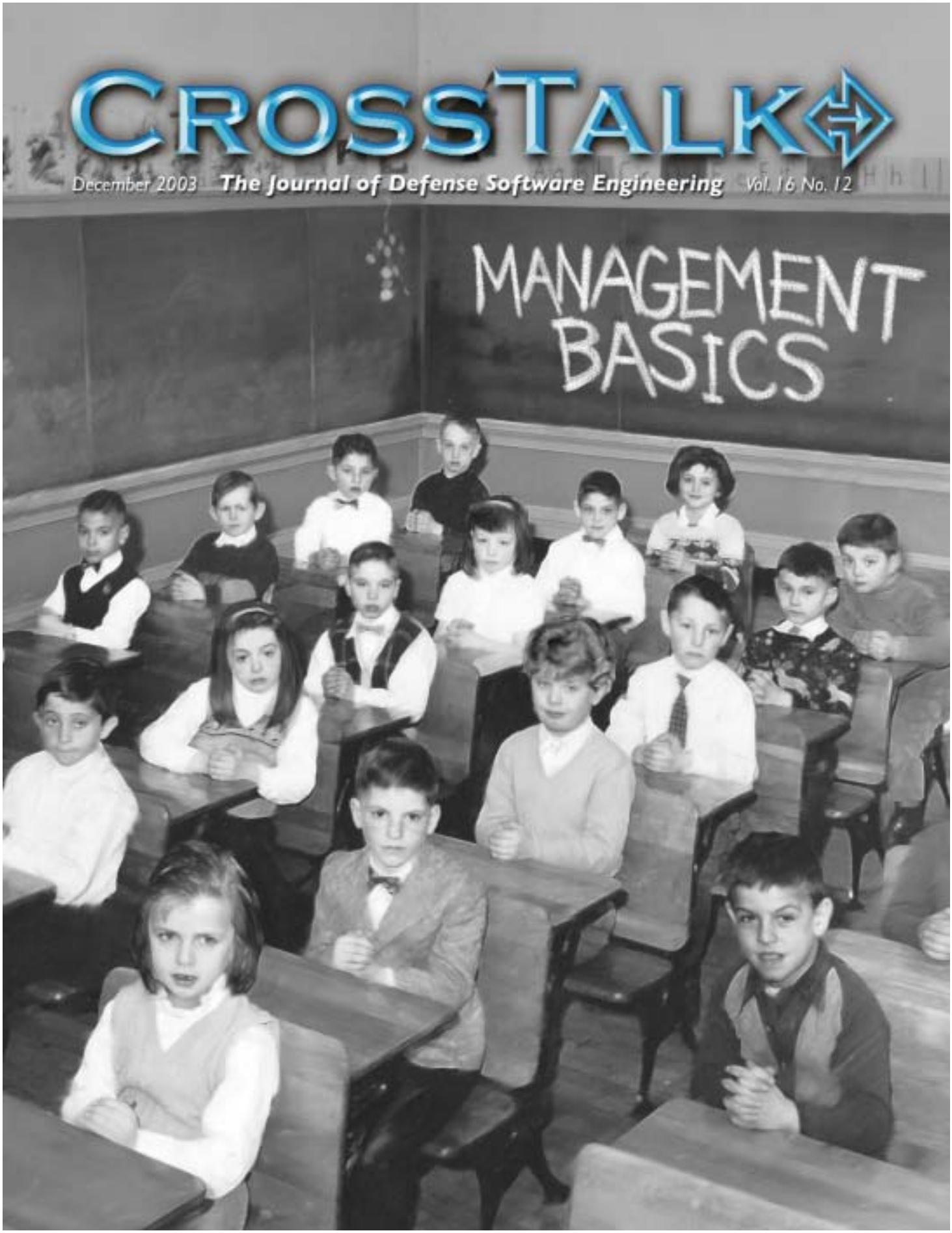
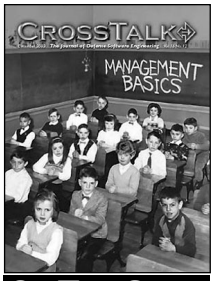
CrossTalk

December 2003 The Journal of Defense Software Engineering Vol. 16 No. 12



Management Basics



ON THE COVER
Cover Design by Kent Bingham.

Open Forum

Departments

CROSSTALK

PUBLISHER	*Tracy Stauder*
ASSOCIATE PUBLISHER	*Elizabeth Starrett*
MANAGING EDITOR	*Pamela Palmer*
ASSOCIATE EDITOR	*Chelene Fortier-Lozancich*
ARTICLE COORDINATOR	*Nicole Kentta*
CREATIVE SERVICES COORDINATOR	*Janna Kay Jensen*
PHONE	(801) 586-0095
FAX	(801) 777-8069
E-MAIL	crosstalk.staff@hill.af.mil
CROSSTALK ONLINE	www.stsc.hill.af.mil/crosstalk

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 28.

Ogden ALC/MASE
6022 Fir Ave.
Bldg. 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

Reprints and Permissions: Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request with CROSSTALK.

Trademarks and Endorsements: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

Coming Events: We often list conferences, seminars, symposiums, etc. that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CROSSTALK Editorial Department.

STSC Online Services: www.stsc.hill.af.mil
Call (801) 777-7026, e-mail: randy.schreifels@hill.af.mil

Back Issues Available: The STSC sometimes has extra copies of back issues of CROSSTALK available free of charge.

The Software Technology Support Center was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

Management Basics: A Necessary Foundation

If you look back at the managers you worked for in your career, I'm sure the names of those who you considered quite good at their jobs come to mind quickly. Moreover, I'm sure the names of those that were not so good also come to mind. Now that I am a manager, I often try to remember what made the good managers in my career so good. How can I learn from them? All managers want to be good managers, but how do they ensure that they are doing their best and meeting their employees' needs and expectations? This month's issue of CROSSTALK highlights management basics. To do their best, I believe that managers need to conquer the *basics* first and then continually improve upon a sound foundation of management principles.

We begin our issue with *People Factors in Software Management: Lessons From Comparing Agile and Plan-Driven Methods* by Richard Turner and Barry Boehm. In this article, the authors address five basic, people-related management areas: staffing, culture, values, communications, and expectations management. Whether developing software under agile, plan-driven, or hybrid methods, the authors emphasize that managers improving upon these five key areas are more likely to succeed at software project management.

Next, in *Back to the Basics: Measurement and Metrics* by Tim Perkins, Roald Peterson, and Larry Smith, we are reminded of the importance of a measurement program as a basic, decision-making tool for managers. The authors have included a Measurement and Metrics Checklist to assist managers in developing, implementing, and reviewing their metrics programs.

Our Management Basics section continues with *How to Talk About Work Performance: A Feedback Primer* by Esther Derby. One of the most uncomfortable tasks for a manager is to provide criticism to an employee who is not performing well. Providing feedback – whether the news is good or bad for the employee – is something every manager must do. This author presents good advice and 11 feedback guidelines to follow to make feedback effective. Next, Johanna Rothman of Rothman Consulting Group, Inc. writes of her 15 years of management experience in *Successful Software Management: 14 Lessons Learned*. This author shares management lessons she has learned while balancing business, employee, and work environment needs.

Wrapping up our theme section is *Deciding to Act* by Walt Lipke. Project managers focus much of their time on monitoring their project's performance in terms of cost, schedule, and quality. This author presents an approach for project analysis and decision-making to assist a project manager on knowing when to act if performance begins to weaken, and what action should be taken to strengthen project performance.

Our Open Forum article this month is *Requirements Engineering Maturity in the CMMI* by Dennis Linscomb. This author expresses his opinion on the deficiency of the Capability Maturity Model® Integration (CMMI®) in defining requirements engineering (RE) maturity and shares his ideas on how the CMMI model should be revamped in the RE area. If you would like to comment on this author's opinion, drop us a line or a Letter to the Editor at <stsc.customerservice@hill.af.mil>.

Another calendar year for CROSSTALK is wrapping up. Thanks to all of our authors for their fine contributions and to the many readers who continue to turn to our publication as a source for defense software engineering best practices and lessons learned. CROSSTALK's 2003 Article Index can be found on page 29. All of our articles and back issues can be found on our Web site at <www.stsc.hill.af.mil/crosstalk>.

On behalf of the CROSSTALK staff, I wish you a safe and very happy holiday season.

Tracy L. Stauder

Tracy L. Stauder
Publisher

People Factors in Software Management: Lessons From Comparing Agile and Plan-Driven Methods

Richard Turner
The George Washington University

Barry Boehm
University of Southern California

While methodologies, management techniques, and technical approaches are valuable, a study of agile and plan-driven approaches has confirmed that the most critical success factors are much more likely to be in the realm of people factors. This paper discusses five areas where people issues can have a significant impact: staffing, culture, values, communications, and expectations management.

Recently we have been studying the characteristics of agile and plan-driven methods to provide guidance in balancing the agility and discipline required for successful software acquisitions or developments [1]. One of the most significant results of our analysis was the realization that, while methodologies, management techniques, and technical approaches are valuable, the most critical success factors are much more likely to be in the realm of people factors.

We believe that the agilists have it right in valuing individuals and interactions over processes and tools [2]. However, they are not the first to emphasize this. There is a long list of wake-up calls: Weinberg's 1971 "Psychology of Computer Programming" [3], the Scandinavian participatory design movement [4], DeMarco and Lister's 1987 "Peopleware" [5], and Curtis' studies of people factors [6] and development of the People Capability Maturity Model® [7].

There is also a wealth of corroborative evidence that people factors dominate other software cost and quality drivers. These include the 1986 Grant-Sackman experiments showing 26:1 the variations in people's performance [8], and the 1981 and 2000 Constructive Cost Model (COCOMO) and COCOMO II cost model calibrations showing 10:1 the effects of personnel capability, experience, and continuity [9, 10]. However, the agilists may finally provide a critical mass of voices amplifying this message.

In this article, we discuss five areas where we believe significant progress can be made: staffing, culture, values, communications, and expectations management.

Staffing

In essence, software engineering is done "of the people, by the people, and for the people."

- **Of the People.** People organize themselves into teams to develop mutually satisfactory software systems.
- **By the People.** People identify what software capabilities they need, and other people develop these for them.
- **For the People.** People pay the bills for software development and use the resulting products.

The two primary categories of players in the software development world are customers and developers.

Customers

Unfortunately, software engineering is still struggling with a *separation-of-concerns* legacy that contends translating customer requirements into code is so hard that it must be accomplished in isolation from people concerns – even the customer's. A few quotes will illustrate the situation:

- The notion of *user* cannot be precisely defined, and therefore it has no place in computer science or software engineering [11].
- Analysis and allocation of the system requirements is not the responsibility of the software engineering group, but it is a prerequisite for their work [12].
- Software engineering is not project management [13].

In today's and tomorrow's world, where software decisions increasingly drive system outcomes, this separation of concerns is increasingly harmful. Customers must be more closely related to the development process. One of the major differences between agile and plan-driven methods is that agile methods strongly emphasize having dedicated and collocated customer representatives, while plan-driven methods count on a good deal of up-front, customer-developer work on contractual plans and specifications.

For agile methods, the greatest risk is that insistence on a dedicated, collocated customer representative will cause the customer organization to supply the person that is most expendable. This risk establishes the need for criteria to determine the adequacy of customer representatives.

In our critical success factor analysis of more than 100 e-services projects at the University of Southern California, we have found that success depends on having customer representatives who are collaborative, representative, authorized, committed, and knowledgeable (CRACK) performers. If the customer representatives are not collaborative, they will sow discord and frustration, resulting in the loss of team morale. If they are not representative, they will lead the developers to deliver unacceptable products. If they are not authorized, they will incur delays seeking authorization or, even worse, lead the project astray by making unauthorized commitments. If they are not committed, they will not do the necessary homework and will not be there when the developers need them most. Finally, if they are not knowledgeable, they will cause delays, unacceptable products, or both.

This summary of customer impact on the landmark Chrysler Comprehensive Compensation project, considered to be the first eXtreme Programming (XP) project, is a good example of the need for CRACK customer representatives.

> The on-site customer in this project had a vision of the perfect system she wanted to develop. She was able to provide user stories that were easy to estimate. Moreover, she was with the development team every day, answering any business questions the developer had.
>
> Halfway [through] the project, several things changed, which eventually led to the project being cancelled. One of the changes was the replacement of the on-site customer, showing that the actual way in which the customer is involved is one of the key success factors in an XP project. The new on-site customer was present most of the time, just like the previous on-site customer, and available to the development team for questions. Unfortunately, the requirements

® Capability Maturity Model is registered in the U.S. Patent and Trademark Office.

> and user stories were not as crisp as they were before. [14]

Plan-driven methods also need CRACK customer representatives and benefit from full-time, on-site participation. Good planning artifacts, however, enable them to settle for part-time CRACK representatives who provide further benefits by keeping active in customer operations. The greatest customer challenge for plan-driven methods is to keep project control from falling into the hands of overly bureaucratic contract managers who prioritize contract compliance above getting project results.

A classic example of customer bureaucracy is provided in Robert Britcher's book, "The Limits of Software" [15], describing his experience on perhaps the world's biggest failed software project: the FAA/IBM Advanced Automation System for U.S. National Air Traffic Control. Due to many bureaucratic and other problems, including responding to change over following a plan, the project was overrun by several years and billions of dollars.

For example, one of the software development groups came up with a way to reduce the project's commitment to a heavyweight brand of software inspections that were slowing the project down by consuming too much staff effort in paperwork and redundant tasks. The group devised a lightweight version of the inspection process. It was comparably successful in finding defects, but with much less time and effort. Was the group rewarded for doing this? No. The contracting bureaucracy sent them a cease-and-desist letter faulting them for contract noncompliance and ordered them to go back to the heavyweight inspections. Agilists justifiably deride this kind of plan-driven bureaucracy.

Developers

Critical people-factors for developers using agile methods include amicability, talent, skill, and communication [16]. An independent assessment identifies this as a potential problem for agile methods: "There are only so many Kent Becks in the world to lead the team. All of the agile methods put a premium on having premium people …" [17]. Figure 1 distinguishes the most effective operating points of agile and plan-driven projects [18, 19]. Both operate best with a mix of developer skills and understanding, but agile methods tend to need a richer mix of higher-skilled people.

When you have such people available on your project, statements like, "A few designers sitting together can produce a better design than each could produce alone," are valid. If not, you are more likely to get design by committee, with the opposite effect. The plan-driven methods do better with great people, but are generally more able to plan the project and architect the software so that less-capable people can contribute with low risk. A significant consideration here is the unavoidable statistic that 49.999 percent of the world's software developers are below average (slightly more precisely, below median).

It is important to be able to classify the type of personnel required for success in the various methods. Alistair Cockburn has addressed levels of skill and understanding required for performing various method-related functions, such as using, tailoring, adapting, or revising a method. Drawing on the three levels of understanding in the martial art Aikido, he has identified three levels of software method understanding that help sort out what various levels of people can be expected to do within a given method framework [18]. Modifying his work to meet our needs, we split his Level 1 to address some distinctions between agile and plan-driven methods, and added an additional level to address the problem of method-disrupters. Our version is provided in Table 1.

The characteristics of Level -1 people should be rapidly identified and reassigned to work other than performing on either agile or plan-driven teams; we recommend such activities as commercial off-the-shelf assessment or *my-four-year-old-can't-break-it* testing.

Level 1B people are average and below, less-experienced, hard-working developers. They can function well in performing straightforward software development in a stable situation. However, they are likely to slow an agile team trying to cope with rapid change, particularly if they form a majority of the team. They can form a well-performing majority of a stable, well-structured, plan-driven team.

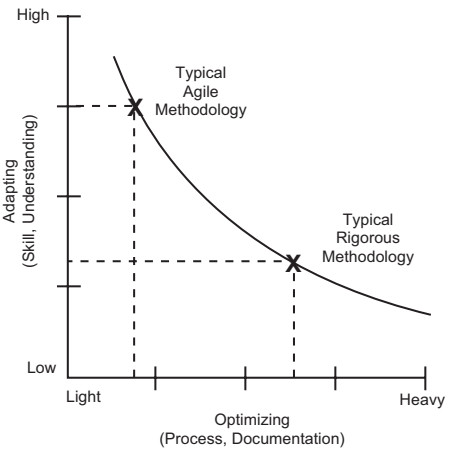


Figure 1: *Balancing, Optimizing, and Adapting Dimensions* [16]

Level 1A people can function well on agile or plan-driven teams if there are enough Level 2 people to guide them. When agilists refer to being able to succeed on agile teams with a ratio of five Level 1 people per each Level 2 person, they are generally referring to Level 1A people.

Level 2 people can function well in managing a small, precedented agile or plan-driven project but need the guidance of Level 3 people on a large or unprecedented project. Some Level 2s have the capability to become Level 3s with experience. Some do not.

Staffing and the Home Grounds

We found that these skill levels were one of the five key discriminators in determining whether a new project would best fit the *home grounds* of agile and plan-driven methods. Home grounds are the set of conditions under which the methods are most likely to succeed. In Figure 2 (see page 6), we graphically portray these home grounds based on five critical factors. In general, the closer to the center, the more the factors favor agility.

The *personnel* axis in Figure 2 shows that the home ground for agile methods requires at least 30 percent to 35 percent

Table 1: *Levels of Software Method Understanding and Use* [17]

Level	Characteristics
3	Is able to revise a method (break its rules) to fit an unprecedented new situation.
2	Is able to tailor a method to fit a precedented new situation.
1A	With training, is able to perform discretionary method steps (e.g., sizing stories to fit increments, composing patterns, compound refactoring, and complex COTS integration). Can become Level 2 with experience.
1B	With training, is able to perform procedural method steps (e.g., coding a simple method, simple refactoring, following coding standards and capability model procedures, and running tests). Can master some Level 1A skills with experience.
-1	May have technical skills, but is unable or unwilling to collaborate or follow shared methods.

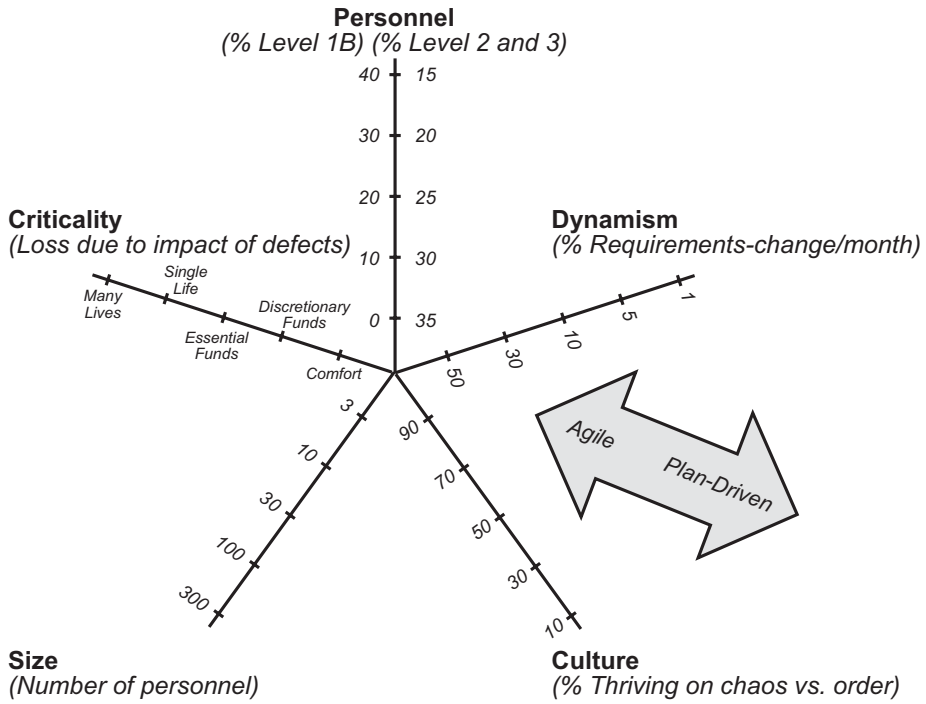


Figure 2: *Key Discriminators of Agile and Plan-Driven Home Grounds*

of the project's people to have Level 2 and 3 skills, with no more than 10 percent of the people with Level 1B skills. The home ground for plan-driven methods can succeed with up to 30 percent to 40 percent Level 1B people, and as few as 15 percent to 20 percent Level 2 and 3 people.

In fact, three of the five key discriminators in Figure 2 are people-related: *personnel* (as discussed above), *size*, and *culture*. The size of the project is measured in the number of people. Agile methods succeed best on projects of 10 people or less, while plan-driven methods work better on projects of 100 people and up. In his landmark XP book, Kent Beck says,

> Size clearly matters. You probably couldn't run an XP project with a hundred programmers. Not fifty. Nor twenty, probably. Ten is definitely doable. [20]

Projects in the middle range of the key discriminator factors need a hybrid mix of agile and plan-driven methods [1]. We will next look more closely at culture.

Culture

The second area of people possibilities, and the third people-related key discriminator between agile and plan-driven home grounds, is culture. In an agile culture, the people feel comfortable and are empowered when there are *many degrees of freedom* available for them to define and work problems. This is the classic craftsman environment, where each person is expected and trusted to do whatever work is necessary for the success of the project. This includes looking for common or unnoticed tasks and completing them.

In a plan-driven culture, the people feel comfortable and empowered when there are *clear policies and procedures* that define their role in the enterprise. This is more of a production-line environment where each person's tasks are well defined. The expectation is that they will accomplish the tasks to specification so that their work products will easily integrate into others' work products with limited knowledge of what others are actually doing.

These cultures are reinforced as people tend to self-select for their preferred culture, and as people within the culture are promoted to higher management levels. Once a culture is well established, it is difficult and time consuming to change. This cultural inertia may be the most significant challenge to the integration of agile and plan-driven approaches.

To date, agile cultural change has had a bottom-up, revolutionary flavor. Failing projects with no hope of success have been the usual pilots, supported by an *it can't hurt* attitude from management and a *no challenge is too hard* adrenaline-charged response from practitioners. Successes have been extraordinary in many cases and have been used to defend migration to less troubled projects.

Early Capability Maturity Model® for Software (SW-CMM®) [21] adopters faced similar challenges, although there was early involvement of middle management. The concept of culture change evolved rapidly and is now well understood by the managers and software engineering process groups. These have been the main change agents in evolving their organizations from following improvised, ad hoc processes toward following plan-driven, SW-CMM-compliant processes.

The new CMM Integration℠ (CMMI®) [22] upgrades the SW-CMM in more agile directions, with new process areas for integrated teaming, risk management, and overall integrated systems and software engineering. A number of organizations are welcoming this opportunity to add more agility to their organizational culture. Others that retain a more bureaucratic interpretation of the SW-CMM are facing the challenge of *change-averse change agents* who have become quite comfortable in their bureaucratic culture.

Values

Along with people come values – different values. One of the most significant and underemphasized challenges in software engineering is to reconcile different users', customers', developers', and other success-critical stakeholders' value propositions about a proposed software system into a mutually satisfactory win-win system definition and outcome. Unfortunately, software engineering is caught in a value-neutral time warp, where every requirement, use case, object, test case, and defect is considered to be equally important.

Most process improvement initiatives and debates, including the silver-bullet debate are inwardly focused on improving software productivity rather than outwardly focused on delivering higher value per unit cost to stakeholders. Again, agile methods and their attention to prioritizing requirements and responding to changes in stakeholder value propositions are pushing us in more high-payoff directions.

Other aspects of value-based software engineering practices and payoffs are described in "Value-Based Software Engineering" [23]. These include the DMR Consulting Group's Benefits Realization Approach and Results Chains [24], stakeholder win-win requirements negotiation [25], business case analysis [26], and the Kaplan-Norton Balanced Scorecard technique [27].

Communications

Even with closely knit in-house develop-

℠ CMM Integration is a service mark of Carnegie Mellon University.

® CMM is registered in the U.S. Patent and Trademark Office.

ment organizations, the "I can't express exactly what I need, but *I'll know it when I see it*" syndrome limits people's ability to communicate an advance set of requirements for a software system. If software definition and development occurs across organizational boundaries, even more communications work is needed to define and evolve a shared system vision and development strategy. The increasingly rapid pace of change exacerbates the problem and raises the stakes of inadequate communication.

Agile methods rely heavily on communication through *tacit*, *interpersonal knowledge* for their success. They cultivate the development and use of tacit knowledge, depending on the understanding and experience of the people doing the work and their willingness to share it. Knowledge is specifically gathered through team planning and project reviews (an activity agilists refer to as *retrospection*). It is shared across the organization as experienced people work on more tasks with different people.

Agile methods generally exhibit more frequent, person-to-person communication. As stated in the Agile Manifesto, emphasis is given to *individuals and interactions*. Few of the agile communications channels are one-way, showing a preference for collaboration. Stand-up meetings, pair programming, and the planning game are all examples of the agile communication style and its investments in developing shared tacit knowledge.

Relying completely on tacit knowledge is like performing without a safety net. While things go well, you avoid the extra baggage and setup effort, but there may be situations that will make you wish for that net. Assuming that everyone's tacit knowledge is consistent across a large team is risky, and as people start rotating off the team, the risk gets higher.

At some point, a group's ability to function exclusively on tacit knowledge will run up against well-known scalability laws for group communication. For a team with N members, there are N(N-1)/2 different interpersonal communication paths to keep current. Even broadcast techniques such as stand-up group meetings and hierarchical team-of-teams techniques run into serious scalability problems.

Plan-driven methods rely heavily on *explicit documented knowledge*. With plan-driven methods, communication tends to be one-way. Communication is generally from one entity to another rather than between two entities. Process descriptions, progress reports, and the like are nearly always communicated as unidirectional flow.

We should note that this distinction between *agile-tacit* and *plan-driven-explicit* is not absolute. Agile methods' source code and test cases certainly qualify as explicit documented knowledge, and even the most rigorous plan-driven method does not try to get along without some interpersonal communication to ensure a consistent, shared understanding of documentation intent and semantics.

When agile methods employ documentation, they emphasize doing the minimum essential amount. Unfortunately, most plan-driven methods suffer from a *tailoring-down* syndrome, which is sadly reinforced by most government procurement regulations. These plan-driven methods are developed by experts who want them to provide users with guidance for most or all foreseeable situations. The experts, therefore, make them very comprehensive, but tailored down for less critical or less complex situations. The experts understand tailoring the methods and often provide guidelines and examples for others to use.

Unfortunately, less expert and less self-confident developers, customers, and managers tend to see the full-up set of plans, specifications, and standards as a security blanket. At this point, a sort of Gresham's Law (*bad money drives out good money*) takes over, and the least-expert participant can drive the project by requiring the full set of documents rather than an appropriate subset. While the nonexperts rarely read the ever-growing stack of documents, they will maintain a false sense of security in the knowledge they have followed best practices to ensure project predictability and control. Needless to say, the expert methodologists are then frustrated with how their tailorable methods are used – and usually verbally abused – by developers and acquirers alike. Agilists have certainly highlighted this significant problem in plan-driven methods.

Except for the landmark people-oriented sources mentioned above, there are frustratingly few sources of guidance and insight on what kinds of communications work best in what situations. Cockburn's "Agile Software Development" [18] is a particularly valuable recent source. It gets its priorities right by not discussing methods until the fourth chapter, and spending the first hundred or so pages discussing why we have problems communicating, and what can be done about it. It nicely characterizes software development as a cooperative game of invention and communication, and provides numerous helpful communication concepts and techniques. Some examples are his definition of the three skill levels based on Aikido discussed earlier, human success and failure modes, information radiators and convection currents, and the effects of distance on communication effectiveness.

Expectations Management

Our primary conclusion in analyzing software project critical success factors has been that the differences between successful and troubled software projects are most often the difference between good and bad expectations management. This coincides with a major finding in a recent root-cause analysis of trouble factors in Department of Defense software projects [28].

Most software people do not do well at expectations management. They have a strong desire to please and to avoid confrontation, and have little confidence in their ability to predict software project schedules and budgets, making them a pushover for aggressive customers and managers trying to get more software for less time and money.

The most significant factor in successful agile or plan-driven teams is that they have enough process mastery, preparation, and courage to be able to get their customers to agree to reduce functionality or increase schedule in return for accommodating a new high-priority change. They are aware that setting up unrealistic expectations is not a win for the customers either, and are able to convince the customers to scale back their expectations. Both agile short iterations and plan-driven productivity calibration are keys to successfully managing software expectations.

Conclusion

Giving top-priority attention to such people-related factors as staffing, culture, values, communications, and expectations management is critical to successful software development and management. Beyond this top-level summary of key factors, there are many valuable sources of guidance on how to succeed with the people-related aspects of your software projects.

Besides the classic Weinberg, Ehn, and DeMarco-Lister books previously cited, there are some further references that can help you improve your people factors whether you use agile, plan-driven, or hybrid development approaches. Good agilist treatments of people and their ecosystems are provided in Jim Highsmith's "Agile Software Development Ecosystems" [19] and Alistair Cockburn's "Agile Software Development" [18]. Complementary plan-driven approaches are provided in Watts

Humphrey's "Managing Technical People" [29] and his Personal Software Process℠ [30], as well as the People CMM developed by Bill Curtis, Bill Hefley, and Sally Miller [31].

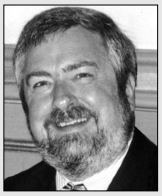
As engineers, our selection of reading materials tends to gravitate toward programming, architecture, or processes for our next learning experience. We strongly recommend you choose one of the books above as a way to balance your technical and people skills.◆

References

1. Boehm, B., and R. Turner. Balancing Agility and Discipline: A Guide for the Perplexed. Boston: Addison-Wesley, 2004.
2. Beck, K., et al. "The Agile Manifesto." The Agile Alliance, 2001 <www.agile alliance.com>.
3. Weinberg, G. The Psychology of Computer Programming. New York: Van Nostrand-Reinhold, 1971.
4. Ehn, P., Ed. Work-Oriented Design of Computer Artifacts. Mahwah, NJ: Lawrence Earlbaum Associates, Mar. 1990.
5. DeMarco, T., and T. Lister. Peopleware: Productive Projects and Teams. New York: Dorset House, 1999.
6. Curtis, B., H. Krasner, and N. Iscoe. "A Field Study of the Software Design Process for Large Systems." Comm. ACM 31. 11 (Nov. 1988): 1268-1287.
7. Curtis, B. et al. People Capability Maturity Model. Reading, MA: Addison-Wesley, 2001.
8. Grant, E., and H. Sackman. "An Exploratory Investigation of Programmer Performance Under On-Line and Off-Line Conditions." Report SP-2581, System Development Corp., Sept. 1966.
9. Boehm, B. Software Engineering Economics. Upper Saddle River, NJ: Prentice Hall, 1981.
10. Boehm, B., et al. Software Cost Estimation With COCOMO II. Upper Saddle River, NJ: Prentice Hall, 2000.
11. Dijkstra, E. Panel Discussion. Fourth International Conference on Software Engineering, 1979.
12. Paulk, M., et al. The Capability Maturity Model for Software: Guidelines for Improving the Software Process. Reading, MA: Addison-Wesley, 1994.
13. Tucker, A. "On the Balance Between Theory and Practice." IEEE Software Sept.-Oct. 2002.
14. van Duersen, A. "Customer Involvement in Extreme Programming." ACM Software Engineering Notes Nov. 2001: 70-73.
15. Britcher, R. N. The Limits of Software. Reading, MA: Addison-Wesley, 1999.
16. Highsmith, J., and A. Cockburn. "Agile Software Development: The Business of Innovation." Computer Sept. 2001: 120-122
17. Constantine, L. "Methodological Agility." Software Development June 2001: 67-69.
18. Cockburn, A. Agile Software Development. Boston: Addison-Wesley, 2002.
19. Highsmith, J. Agile Software Development Ecosystems. Boston: Addison-Wesley, 2002.
20. Beck, K. Extreme Programming Explained. Boston: Addison-Wesley, 1999: 157.
21. Paulk, M., et al. The Capability Maturity Model. Reading, MA: Addison-Wesley, 1994.
22. Ahern, D. M., A. Clouse, and R. Turner. CMMI Distilled: A Practical Introduction to Integrated Process Improvement. 2nd ed. Boston: Addison-Wesley, 2003.
23. Boehm, B. "Value-Based Software Engineering." ACM Software Engineering Notes Mar. 2003.
24. Thorp, J. The Information Paradox. McGraw-Hill, 1998.
25. Boehm, B., P. Bose, E. Horowitz, and M. J. Lee. Software Requirements as Negotiated Win Conditions. Proc. of the First International Conference on Requirements Engineering, Colorado Springs, CO. IEEE Computer Society Press, Apr. 1994.
26. Reifer, D. Making the Software Business Case. Boston: Addison-Wesley, 2002.
27. Kaplan, R., and D. Norton. The Balanced Scorecard: Translating Strategy into Action. Boston: Harvard Business School Press, 1996.
28. McGarry, J., and Charette, R. "Systemic Analysis of Assessment Results from DoD Software-Intensive System Acquisitions." Tri-Service Assessment Initiative Report, Office of the Under Secretary of Defense (Acquisition, Technology, Logistics), 2003.
29. Humphrey, W. Managing Technical People. Boston: Addison-Wesley, 1997.
30. Humphrey, W. Introduction to the Personal Software Process. Boston: Addison-Wesley, 1997.
31. Curtis, B., B. Hefley, and S. Miller. The People Capability Maturity Model. Boston: Addison-Wesley, 2001.

℠ Personal Software Process is a service mark of Carnegie Mellon University.

About the Authors



Richard Turner, D.Sc., is a member of the Engineering Management and Systems Engineering Faculty at The George Washington University in Washington, D.C. Currently, he is the assistant deputy director for Software Engineering and Acquisition in the Software Intensive Systems Office of the Under Secretary of Defense (Acquisition, Technology, and Logistics). Turner is co-author of the book "CMMI Distilled."

1931 Jefferson Davis Highway
Suite 104
Arlington, VA 22202
Phone: (703) 602-0581 ext. 124
E-mail: rich.turner.ctr@osd.mil

Barry Boehm, Ph.D., is the TRW professor of software engineering and director of the Center for Software Engineering at the University of Southern California. He was previously in technical and management positions at General Dynamics, Rand Corp., TRW, and the Office of the Secretary of Defense as the director of Defense Research and Engineering Software and Computer Technology Office. Boehm originated the spiral model, the Constructive Cost Model, and the stakeholder win-win approach to software management and requirements negotiation.

University of Southern California
Center for Software Engineering
Los Angeles, CA 900989-0781
Phone: (213) 740-8163
(213) 740-5703
Fax: (213) 740-4927
E-mail: boehm@sunset.usc.edu

Back to the Basics: Measurement and Metrics

Tim Perkins and Roald Peterson
Software Technology Support Center/SAIC

Larry Smith
Software Technology Support Center

Measurements and metrics are key tools to understanding the behaviors, successes, and failures of our programs and projects. This article highlights the basic principles of measures and metrics and encourages the reader to improve his or her use of these tools. The article is adapted from [1].

According to Tom DeMarco, "You cannot control what you cannot measure" [2]. Imagine going on a road trip of over a thousand miles. This is easy because most of us really have done this several times. Now imagine that your car has no speedometer, no odometer, no fuel gauge, and no temperature indicator. Imagine also that someone has removed the mile markers and road signs from all the roads between you and your destination. Just to complete the experiment, remove your watch.

What was once a simple journey becomes an endless series of guesses, fraught with risks. How do you know where you are, how far you have gone, or how far you have to go? When do you gas the car? Should you stop here or try to make the next town before nightfall? You could break down, run out of gas, be stranded, take the wrong road, bypass your destination, or waste time trying to find your location and how to reach your destination. Clearly, some method of measuring certain indicators of progress is essential for achieving a goal.

Imagine again going on a road trip. This time the cockpit of the car is filled with instruments. In addition to what you have been accustomed to in the past, there are now the following gauges:

- Speed in feet and yards per second, and as a percentage of *c* (light speed).
- Oil pressure in millibars.
- Estimated time to deplete or recharge the battery.
- Fuel burn rate and fuel weight.
- Oil viscosity and transparency indicators.
- Antifreeze temperature and pressure.
- Engine efficiency.
- Air conditioning system parameters (pressures, temperatures, efficiency).
- Elevation, rate of climb, heading, accelerometers for all directions.
- Indicators for distance and time to destination and from origin.
- Inside air temperatures for eight different locations in the car.

Also, there are instruments to count how many cars pass, vibration levels, and sound pressure levels within and outside the car. There are weather indicators for outside temperature, humidity, visibility, cloud ceiling, ambient light level, true and relative wind speeds and directions, warning indicators for approaching storms and seismic activity, etc. Along the roads will be markers for every hundredth mile and signs announcing exits every quarter mile for five miles before an exit is reached. Signs in five-mile-per-hour increments will announce speed changes.

"Metrics are measurements of different aspects of an endeavor that help us determine whether we are progressing toward the goal of that endeavor."

To some, this may seem like a dream come true, at least the cockpit part. However, careful consideration will soon reveal that the driver will be inundated and quickly overwhelmed with unnecessary, confusing data. Measurement, in itself, is no prescription for achieving a goal. It can even make the goal unattainable.

Introduction

Metrics are measurements of different aspects of an endeavor that help us determine whether we are progressing toward the goal of that endeavor. They are used extensively as management tools to provide some calculated, observable basis for making decisions. Some common metrics for projects include schedule deviation, remaining budget and expenditure rate, presence or absence of specific types of problems, and milestones achieved. Without some way to accurately track budget, time, and work progress, a project manager can only make decisions in the dark. Without a way to track errors and development progress, software development managers cannot make meaningful improvements in their processes. The more inadequate our metrics program, the closer we are to herding black cats in a dark room. The right metrics, used in the right way, are absolutely essential for project success.

Too many metrics are used simply because they have been used for years, and people believe they might be useful [3]. Each metric should have a purpose, providing support to a specific decision-making process. Leadership too often dictates metrics. A team under the direction of leadership should develop them. Metrics should be used not only by leadership but also by all the various parts of an organization or development team. Obviously, not all metrics that are useful to managers are useful to the accounting people or to developers. Metrics must be tailored to the users. The use of metrics should be defined by a program describing what metrics are needed, by whom, and how they are to be measured and calculated. The level of success or failure of your project will depend in large part on your use or misuse of metrics – on how you plan, implement, and evaluate an overall metrics program.

While this article introduces and describes key metrics ideas and processes and can point you in the right direction, it is recommended that you gain more insight, depth, and specific examples by downloading and reading the material listed in "Additional Reading" in the online version of this article at <www.stsc.hill.af.mil/crosstalk>. Of particular value to Department of Defense users is [4] better known as the "Practical Software and System Measurement Guidebook," where a more specific and detailed terminology and methodology can be found.

Process Description

Metrics are not defined and used solely,

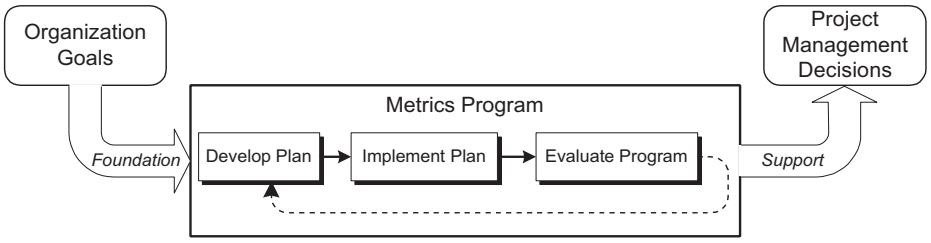


Figure 1: *Metrics Program Cycle*

Figure 2: *Basili's Goal, Question, Metric Paradigm*

but are part of an overall metrics program. This program should be based on the organization's goals and should be carefully planned, implemented, and regularly evaluated for effectiveness. The metrics program is used as a decision support tool.

In relation to project management metrics, if the information provided through a particular metric is not needed for determining status or direction of the project, it is probably not needed at all. Process-related metrics, however, should not necessarily be dismissed so harshly since they indicate data useful in improving performance across repeated applications. The role of the metrics program in the organization and its three major activities are shown in Figure 1.

Developing a Metrics Program Plan

The first activity in developing a metrics program is planning. Metrics planning is usually based on the goal-question-metric (GQM) paradigm developed by Victor Basili (see Figure 2). The GQM paradigm is based on the following key concepts [3]:

1. Processes, including software development, program management, etc., have associated goals.
2. Each goal leads to one or more questions regarding the accomplishment of the goal.
3. Each question leads to one or more metrics needed to answer the question.
4. Each metric requires two or more measurements to produce the metric (e.g., miles per hour, budget spent vs. budget planned, temperature vs. operating limits, actual vs. predicted execution time, etc.).
5. Measurements are selected to provide data that will accurately produce the metric.

The planning process is comprised of the three sub-activities implementing the GQM paradigm and one that defines the data collection process. Each of these is discussed in the following sections.

Table 1 shows two examples of goals and their related questions and metrics. Note that there could be one or more metrics associated with each question. As the initial list of questions and metrics is written and discussed, the goal is usually refined, which then causes a further refinement in the accompanying questions and metrics.

Define Goals

Planning begins with well defined, validated goals. Goals should be chosen and worded in such a way that they are verifiable; that is, their accomplishment can be measured or observed in some way. Goals such as meeting a specific delivery schedule are easily observable. Requirements stating "software shall be of high quality" are highly subjective and need further definition before they can be used as valid goals.

You may have to refine or even derive your own goals from loosely written project objectives. The selection or acceptance of project goals will determine how you manage your project, and where you put your emphasis. Goals should meet the following criteria:

- They should support the successful accomplishment of the project's overall or system-level goals.
- They should be verifiable, or measurable in some way.
- They should be defined in enough detail to be unambiguous.

Derive Questions

Each goal should evoke questions about how its accomplishment can be measured. For example, completing a project within a certain budget may evoke questions such as these: What is my total budget? How much of my budget is left? What is my current spending rate? Am I within the limits of my spending plan?

Goals related to software time, size, quality, or reliability constraints would evoke different questions. It should be remembered that different levels and groups within the organization might require different information to measure the progress in which they are interested. Questions should be carefully selected and refined to support the previously defined project goals. Questions should exhibit the following traits:

- Questions only elicit information that indicates progress toward or completion of a specific goal.
- Questions can be answered by providing specific information. (They are unambiguous.)
- Questions ask all the information needed to determine progress or completion of the goal.

Once questions have been derived that elicit only the complete set of information needed to determine progress, metrics must be developed that will provide that information.

Develop Metrics

Metrics are the information needed to answer the derived questions. Each question can be answered by one or more metrics. These metrics are defined and associated with their appropriate questions and goals. Each metric requires two or more measurements. Measurements are those data that must be collected and analyzed to produce the metric.

Measurements are selected that will provide the necessary information with the least impact to the project workflow.

Table 1: *Goals and Their Related Questions and Metrics*

	Common Example	Product Example
Goal	***Run competitively in a 10 kilometer (10K) race.***	***Improve customer satisfaction with the current release of the product.***
Questions and Metrics	***What is a competitive time for an individual of my age and rank?*** • Age and ranking figures for run times. ***Am I capable of running at a competitive time?*** • Time to complete 10K, post-run recovery time, etc., repeated over each practice run. ***What current injuries are impacting my ability to race?*** • Injury prognosis and recovery time. ***Am I sustaining my health and weight by eating and sleeping properly?*** • Hours of sleep per night, weight, dietary intake, etc.	***Are customers buying our product?*** • Sales rate (up or down) as compared with competing products, product return rate, etc. ***What are the key attributes of customer satisfaction?*** • Metrics related to reliability, safety, functionality, performance, etc. ***How satisfied are our customers (in relation to the above attributes)?*** • Customer survey data, defects reported, etc. ***How are we resolving problems that affect customer satisfaction?*** • Defect resolution rate, post-release defect density, etc.

Figure 3 summarizes the process of turning measurements into goal status.

Choosing measures is a critical and nontrivial step. Measurements that require too much effort or time can be counterproductive and should be avoided. Remember, just because something can be measured does not mean it should be. An in-depth introduction to measurements, "Goal-Driven Software Measurement – A Guidebook," has been published by the Software Engineering Institute and is available as a free download [5].

In addition to choosing what type of data to collect or measure, the methods of processing or analysis must also be defined in this step. How do you turn the measurements into a meaningful metric? How does the metric then answer the question? The analysis method should be carefully documented. Do not assume that it is obvious.

This activity is complete when you know exactly what type of data you are going to collect (what you are going to measure and in what units), how you are going to turn that data into metrics (analysis methods), and in what form (units, charts, colors, etc.) the metrics will be delivered.

Define the Collection Process

The final step of the metrics planning process is to determine how the metrics will be collected. At a minimum, this part of the plan should include the following:

- What data is to be collected?
- What will be the source of the data?
- How is it to be measured?
- Who will perform the measurement?
- How frequently should the data be collected?
- Who will the derived metrics be delivered to, and in what format?

Implementing a Metrics Program

A good rule of thumb to follow when starting a measurement program is to keep the number of measurements between five and 10. If the metrics program is well planned, implementing the program should be reduced to simply following the plan. There are four activities in the metrics implementation cycle, shown in Figure 4 [6].

Data is collected at specific intervals according to the plan. Data is then validated by examining it to ensure it is the result of accurate measurements, and that the data collection is consistent among members of the group if more than one individual is collecting it. In other words, is it being measured in the same way, at the same time, etc.? Once the data is determined to be valid, the metrics are derived by analyzing the data as documented in the metrics program plan. Metrics are then delivered to appropriate individuals and groups for evaluation and decision-making activities. This process is repeated until the project is complete.

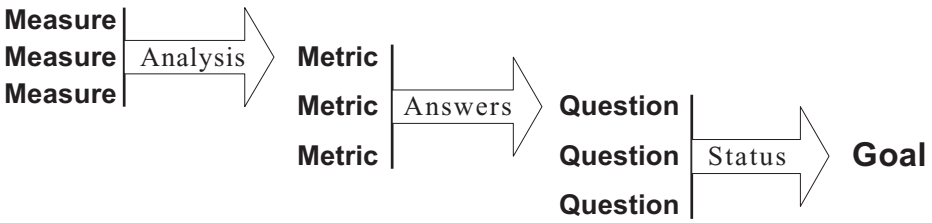


Figure 3: *Goal, Question, Metrics Examples*

Evaluating a Metrics Program

It is likely that a metrics program will not be perfect in its first iteration. Soon after its initial implementation and at regular intervals after that, the metrics program should be evaluated to determine if it is meeting the needs of the metrics users, and if its implementation is flowing smoothly. If metrics prove to be insufficient or superfluous, the program plan should be modified to provide the necessary information and remove any unneeded activity. The objective of a metrics program is to provide sufficient information to support project success while keeping the metrics program as simple and unobtrusive as possible. The following are areas that should be considered when reviewing a metrics program:

- Adequacy of current metrics.
- Superfluity of any metrics or measures.
- Interference of measurements with project work.
- Accuracy of analysis results.
- Data collection intervals.
- Simplification of the metrics program.
- Changes in project or organization goals.

"Too many metrics are used simply because they have been used for years, and people believe they might be useful."

Metrics Repository

A final consideration is establishing a metrics repository where metrics history is kept for future projects. The availability of past metrics data can be a gold mine of information for calibration, planning estimates, benchmarking, process improvement, calculating return on investment, etc. At a minimum, the repository should store the following:

- Description of projects and their objectives.
- Metrics used.
- Reasons for using the various metrics.
- Actual metrics collected over the life of each project.
- Data indicating the effectiveness of the metrics used.

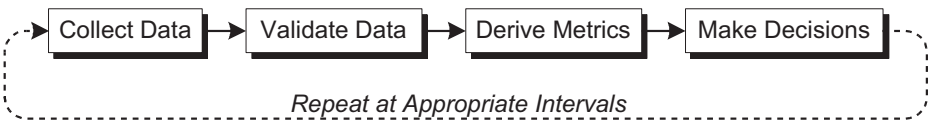
Measurement and Metrics Checklist

This checklist is provided to assist you in developing a metrics program, and in defining and using metrics. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action. The checklist items are divided into three areas: developing, implementing, and reviewing a metrics program.

Developing a Metrics Program

- ☐ Is your use of metrics based on a documented metrics program plan?
- ☐ Are you using the GQM paradigm in developing your metrics?
- ☐ Are your metrics based on measurable or verifiable project goals?
- ☐ Do your goals support the overall system-level goals?
- ☐ Are your goals well defined and unambiguous?
- ☐ Does each question elicit only information that indicates progress toward or completion of a specific goal?

Figure 4: *Metrics Implementation Cycle*



- ☐ Can questions be answered by providing specific information? (Is it unambiguous?)
- ☐ Do the questions ask for all the information needed to determine progress or completion of the goal?
- ☐ Is each metric required for specific decision-making activities?
- ☐ Is each metric derived from two or more measurements (e.g., remaining budget vs. schedule)?
- ☐ Have you documented the analysis methods used to calculate the metrics?
- ☐ Have you defined those measures needed to provide the metrics?
- ☐ Have you defined the collection process (i.e., what, how, who, when, how often, etc.)?

Metrics Program Implementation

- ☐ Does your implementation follow the metrics program plan?
- ☐ Is data collected the same way each time it is collected?
- ☐ Are documented analysis methods followed when calculating metrics?
- ☐ Are metrics delivered in a timely manner to those who need them?
- ☐ Are metrics being used in the decision-making process?

Metrics Program Evaluation

- ☐ Are the metrics sufficient?
- ☐ Are all metrics or measures required, that is, non-superfluous?
- ☐ Are measurements allowing project work to continue without interference?
- ☐ Does the analysis produce accurate results?
- ☐ Is the data collection interval appropriate?
- ☐ Is the metrics program as simple as it can be while remaining adequate?
- ☐ Has the metrics program been modified to adequately accommodate any project or organizational goal changes?◆

References

1. Software Technology Support Center. Condensed Version (4.0) of Guidelines for Successful Acquisition and Management of Software-Intensive Systems. Hill Air Force Base, Utah: Software Technology Support Center, Feb. 2003.
2. DeMarco, Tom. Controlling Software Projects. New York: Yourden Press, 1982.
3. Perkins, Timothy K. "The Nine-Step Metrics Program." CROSSTALK, Feb. 2001 <www.stsc.hill.af.mil/crosstalk/2001/feb/perkins.asp>.
4. Bailey, Elizabeth, et al. Practical Software Measurement: A Foundation for Objective Project Management Ver. 4.0b1. Severna Park, MD: Practical Software and Systems Measurement, Mar. 2003 <www. psmsc.com/PSMGuide.asp> under "Products."
5. Park, Robert E., et al. Goal-Driven Software Measurement – A Guidebook. Pittsburgh, PA: Software Engineering Institute, Aug. 1996 <www.sei.cmu.edu/publications/documents/96.reports/96.hb.002html>.
6. Augustine, Thomas, et al. "An Effective Metrics Process Model." CROSSTALK June 1999 <www.stsc.hill.af.mil/crosstalk/1999/jun/augustine.asp>.

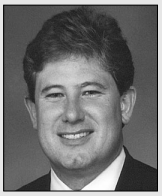
About the Authors

Tim Perkins has been involved in software process improvement for the past 11 years, including leading the effort to initiate software process improvement at the then five Air Force Air Logistics Centers. As the Software Engineering Process Group leader at the Software Engineering Division at Hill Air Force Base, Utah, he led the division in reaching Capability Maturity Model® (CMM®) Level 3. The division has gone on to achieve CMM Level 5. Perkins is Acquisition Professional Development Program Level 3 certified in Project Management and System Planning, Research, Development, and Engineering.

Software Technology Support Center
OO-ALC/MASE
6022 Fir Ave.
Bldg. 1238
Hill AFB, UT 84056
Phone: (801) 775-5736
Fax: (801) 777-8069
E-mail: tim.perkins@hill.af.mil

Roald E. Peterson is a senior systems engineer with Science Applications International Corporation. He has 22 years of electronic systems development experience, specializing in communications, architecture, and software development. Peterson was an editor and contributor for the "Guidelines for the Successful Acquisition and Management (GSAM) of Software Intensive Systems" and is the author of the "Condensed GSAM Handbook." He has a bachelor's degree in physics and master's degrees in computer resources management and electrical engineering.

Science Applications
International Corporation
920 W. Heritage Park Blvd.
Suite 210
Layton, UT 84041
Phone: (801) 774-4705
Fax: (801) 728-0300
E-mail: roald.e.peterson@saic.com



Larry Smith is a senior software engineer and project manager for the Air Force's Software Technology Support Center at Hill Air Force Base, Utah. He provides software engineering, software process improvement, and project management consulting for the U. S. Air Force and other Department of Defense organizations as well as commercial and nonprofit organizations. Smith is a faculty member at the University of Phoenix. He is also certified by the Project Management Institute as a Project Management Professional. Smith has a bachelor's degree in electrical engineering and a master's degree in computer science.

Software Technology Support Center
OO-ALC/MASE
6022 Fir Ave.
Bldg. 1238
Hill AFB, UT 84056
Phone: (801) 777-9712
Fax: (801) 777-8069
E-mail: larry.smith4@hill.af.mil

How to Talk About Work Performance: A Feedback Primer©

Esther Derby
Esther Derby Associates, Inc.

Providing feedback on work performance is part of every manager's job. Feedback is how people know what they are doing well, and what they need to do differently. Unfortunately, many managers do not receive much training on how to give feedback. Managers who are uncomfortable giving feedback may put it off or hope that hints and general statements will make their point. The author shares what she has learned about providing effective feedback, and advises how to get back on track when a feedback receiver has a puzzling response.

In a recent workshop, Alex, a new manager, described a situation involving Marie, one of the people in his group. Marie was normally quiet, but when she felt nervous, she interrupted people. In a recent client meeting, Marie interrupted a key customer four times. Alex could see the client was becoming irritated, but Marie did not seem to be aware of what she was doing, or the effect she was having.

I asked him how he had handled the problem with Marie. "Oh, I haven't talked to her about it yet," the new manager replied. "She's basically a good performer. We have a performance review coming up in three months. I'll tell her about it then."

Excellent managers do not wait for the year-end performance cycle to provide feedback. They provide feedback on what is working well and what is not working frequently and to everyone on their staff, not just the underperformers.

Providing useful feedback is not easy. However, it is an important part of a manager's responsibility. In this article, I will share some of what I have learned about how to deliver effective feedback.

What Is Feedback?

Feedback is the information we give others when we want them to start, stop, continue, or change some behavior [1]. Managers provide the people who report to them with information about results and behavior that relates to work and the work environment. Employees need information to know what they are doing well and where they need to make adjustments to be successful. If managers do not tell them, who will?

According to the authors of "What Did You Say? The Art of Giving and Receiving Feedback," feedback is "information about the past delivered in the present which may influence future behavior" [1]. Let us look at this definition in detail:

- **Information.** This is not a judgment or a label, not criticism or praise. First and foremost feedback is information that we can use to understand the current situation and make choices.
- **About past behavior.** Feedback describes some past behavior, something that can be observed, not an interpretation of events.
- **Which may influence future behavior.** Managers give feedback in the hope of changing some aspect of another person's behavior, but of course, that other person has a choice about what to do with the information given.

"Employees need information to know what they are doing well and where they need to make adjustments to be successful."

What Is in Bounds for Management Feedback?

The goal in giving feedback is not to make sure everyone is charming or performs his or her work exactly the way you would. It is to make sure that the people who work for you are productive. Some of that information you provide to improve productivity may be about work results – timeliness or quality of the work produced or the service delivered.

Other times feedback is about the work environment – the personal actions and interactions that affect the group's ability to work together and accomplish results.

It certainly would make a manager's role easier if everyone had perfect interpersonal skills and behaved congruently at all times. However, we do not. When I started as a manager, I had a notion that everyone would *act like an adult*. In my years as manager, I discovered that definitions of *adult* behavior vary greatly. In fact, most people act like adults much of the time, and sometimes they do not. When humans feel weak, vulnerable, or under stress, they often behave in a not very grown-up manner [2].

Providing feedback about interpersonal behavior is a more delicate proposition than talking about tangible results. Nonetheless, it is critical. When interpersonal skills and behaviors affect the work environment, it is a manager's responsibility to address it.

When you prepare to provide feedback, make sure it is about the work, and that it is important. Some things are not worth bringing up. I had a staff member who mispronounced certain common words. I grew up with a grammarian, and the staff member's mispronounciations annoyed me. My annoyance was about my preferences, not his work results. In the business context, it was not important. If it is not important, do not bring it up.

Some things are not your business as a manager, but what those things are depends on the context. If you work for a corporation, it is probably not appropriate to talk about an employee's financial situation. However, in a military setting, you may need to know about certain financial events. Check with your human resources (HR) representative or commanding officer to learn what the boundaries are in your situation *before* you cross the line.

Feedback Guidelines

I have talked about what feedback is, and when it is appropriate for a manager to give feedback. Now what? The following are some guidelines that will help you provide effective feedback.

Provide Feedback as Close to the Event as Reasonably Possible

Do you remember what you had for

lunch yesterday? Like most people, sometimes I do, and sometimes I do not. If someone asks me about an event that happened several weeks or months ago, I may or may not remember depending on the significance the event held for me.

That is why it is important to give feedback as close to the event as possible. People are not likely to remember past events clearly, particularly when they do not have personal significance.

Marie, the employee whose manager, Alex, decided he would wait three months to give her feedback, may not even remember the incident by the time he brings it up. Marie's behavior stood out to Alex, but since it is an ingrained habit, it might not stand out to Marie.

Worse, if Alex waits to tell Marie, how many times will she repeat the gaffe in the intervening three months? Why rob Marie of a chance to improve now rather than later? If Marie's actions affect the entire group by damaging the relationship with the client, why accept the negative effects on the group for longer than necessary?

It might not be easy for Marie to change her habit, but it will be impossible if she is not aware of it.

Late feedback is a lost opportunity. Give the feedback close to the event so that the individual has a choice to change and to improve group performance.

Provide Feedback on a Regular Basis

Regular one-on-one meetings are a great place to provide feedback. Give information for minor course corrections, and discuss what is going well. If you do not have a minor course correction, do not feel like you have to make one up. There is no need to pounce on every little thing. If an employee breaks the build once in a year, or comes in late once in six months, it is not a performance problem [3].

I have a rule of thumb, though, about noticing some positive aspect on a regular basis. Regular recognition for work well done is one of the keys to being a great manager [4]. Even if you do not have regular meetings, find a way to notice and appreciate the work people do *every week*.

People need to know what they are doing well – and should continue doing – as much as they need to know when they are missing the mark.

Give Serious Feedback in a Serious Setting

Hints and off-hand comments in the hallway usually fail to get the message across. If you want an employee to act on your request, be intentional about it. One-on-ones are fine for small course corrections and positive feedback. Schedule a separate private meeting to address situations that involve a serious performance issue.

Address Individual Issues Individually

Some new managers try to avoid the awkwardness of providing feedback by making policy announcements in staff meetings rather than address the individual directly. General policy announcements do not carry the same weight as an individual conversation.

When the general announcement concerns a specific behavior, the results can be even worse. One new supervisor made a point in his weekly staff meeting of reminding everyone not to use the speakerphone to listen to voice mail. Only one person in the group had this habit and she felt publicly embarrassed when her supervisor brought it up in the staff meeting. The other members of the team felt unfairly chastised.

> *"First and foremost feedback is information that we can use to understand the current situation and make choices."*

If it is important enough to bring up, it is important enough to speak directly to the person involved.

Provide Specific Examples

General statements do not provide enough information for people to know what to improve. Labels, such as *you are sloppy*, set up an oppositional dynamic that goes nowhere. Labels get in the way of solving the problem. Rather than label the behavior, describe it.

Whether you are describing results or behavior, be as specific as you can. A tech writer turned in a chapter that had dozens of spelling and grammar errors. Telling the tech writer that her work is poor quality is not specific enough. If you are concerned about accuracy, state what you have seen: "I noticed that in Chapter 1 there are numerous spelling and grammar errors."

Say that one of your testers loses his temper in a bug-fix meeting. Rather than tell him he has a temper problem, relate what you saw and heard. "In the 1 p.m. meeting, you hit the conference table with your fist several times and your face became red." Then you can discuss what responses are appropriate.

The same guideline holds for feedback about what is going well. Give specific examples. Suppose your technical lead did an exceptional job writing up a site-visit report. Saying, "I really liked the report you wrote," is a nice compliment, but it does not tell your technical lead what you liked or what he or she did well. Saying, "I particularly liked the way the topics were prioritized. It made the report very easy to follow," will let your technical lead know that you value a well-organized, easy-to-follow report. Without specific information on what he or she did well, your technical lead might have concluded that you liked the fact the report was printed on pink paper.

Vague references, labels, and guessing games leave employees feeling resentful and confused. Employees are likely to ignore vague feedback. Specific observations give people the information they need to know what result or behavior they need to change.

Do Not Rely on Mind Reading

Do not expect your employees to be mind readers. One employee was surprised when her manager informed her that she was not meeting expectations. "What am I doing wrong?" she asked. "I expect someone at your grade level to know what to do without me having to spell it out," her manager replied.

Another employee was likewise surprised when his manager told him the way he had handled a customer meeting was unacceptable. "What did I do wrong?" he asked. "You know what you did," the manager replied.

Dropping hints does not work, nor does making the employee guess what you mean. If you want to see a change, say what it is.

Check for Agreement on the Data

Once you have given some specific examples of the result or behavior you have observed, obtain agreement on the data. If the feedback receiver does not agree with the data, it is going to be hard to move into problem solving.

The tech writer who turned in work with errors will probably acknowledge that there were spelling errors in Chapter 1. She is not as likely to admit to being sloppy. The tester who hit the table will probably admit that he hit the table, but may disagree that he has an anger-man-

agement problem.

Employees are more likely to accept what you say if it is specific and observable.

Request a Change

If you want a behavior to stop or continue, say that. If you want a change, say that, too. Do not leave the employee guessing what he or she needs to do to correct the situation.

The tech writer's manager might say something like, "I have marked spelling and proofing errors on the copy you gave me. In the future, I'd like you to run spell check and proofread before you turn in your work."

The table-hitting tester's boss might say, "I know you can't control when your face becomes red; hitting the table with your fist is not acceptable."

Engage in Problem Solving

If you want a different result, but have latitude on how that result is achieved, move into problem-solving mode. The tech writer's manager might say, "I was surprised at the number of spelling and proofing errors in the chapters you turned over to me. I expect copy to be clean when I receive it. What are three ways you could make sure the copy is in good shape?"

The tech writer might reply, "Well, I could run spell check, proof it on paper, or ask Jessica do a final check."

When the employee arrives at the solution, it is more likely to fit his style, and is more likely to stick.

Agree on How You Will Follow Up

Sometimes you will need to follow up on the changes you have requested. The tech writer and her manager might agree that he will skim for obvious spelling and grammar errors and if he finds them, he will return the work to her to fix.

Check for Understanding

Saying the words is not enough. When giving feedback is part of your job, you also need to check to make sure the message you sent is the same one the employee heard. Checking for understanding can help correct some of the slippage that normally occurs in conversation.

One manager I know wraps up feedback conversations by saying, "I'm going to check for understanding now. I'd like you to summarize our conversation for me so I'm sure I have been clear."

Charlie told Shanna, a recent college graduate, that the code she had checked in had broken the build three times in a week. He was concerned that the code was brittle and wanted her to do more unit testing. When Charlie asked Shanna to summarize their feedback conversation, Shanna replied, "You are disappointed in me. I need to be more careful."

Charlie was able to correct Shanna's misperception. He was not disappointed in her – actually, he was quite pleased with her work in general. He had noticed a pattern and wanted her to take some additional steps so she and the entire group would be more successful.

For many people, hearing criticism is an emotionally charged situation. When people are in the throes of an emotional response, they often do not hear clearly.

"The goal in giving feedback is not to make sure everyone is charming or performs his or her work exactly the same way you would. It is to make sure that the people who work for you are productive."

As a feedback giver, you do not have control over how someone else will respond to your feedback. However, checking for understanding can clear up some obvious misinterpretations.

Troubleshooting Feedback Conversations

When you are careful and intentional in giving feedback, you greatly increase the chance that you will be successful. You will state what you have observed clearly. The receiver will agree with your observation, you will do some joint problem solving, and agree on a course of action.

In real life, even when you follow all these guidelines, sometimes the feedback receiver will have a response that is surprising or puzzling.

As we saw with Charlie and Shanna, you cannot control how the feedback receiver interprets what you say, how they react emotionally to that interpretation, and how they respond.

When the receiver responds in a puzzling way – perhaps with an angry outburst or with tears – you will need some strategies to bring the feedback conversation back on track.

Check the Data

When people are upset, they often do not hear clearly. Ask the feedback receiver to repeat what he heard and what he saw. When one manager asked a staff member what he heard her say after a puzzling response, he described the way the manager was leaning forward, the tone of her voice, her facial expression – but no words. He was reacting primarily to what he had seen. Once the manager repeated the words, and her staff member heard them, they were able to move forward.

Check for Interpretation

If the words made it through intact but the interaction is still tangled, check how the receiver interpreted your words. We all make meaning of what we hear; in most cases, our interpretation is close enough to allow productive communication. Sometimes our interpretation is way off, and then it helps to check.

Shanna interpreted Charlie's feedback as criticism. Charlie had noticed a pattern and was offering information to help Shanna be more effective. When Charlie asked Shanna to summarize for understanding, he was able to see that Shanna's interpretation was a little off.

Ask What Is Happening

Sometimes a feedback response is still puzzling even after you have checked the data and the interpretation. This may indicate other forces are at work. Perhaps your feedback triggered a memory or association with a painful past event. Perhaps the feedback receiver has a perfection rule, and any comment that indicates his work is not perfect is difficult. As a manager, it is not your job to psychoanalyze. It is your job to try to get the conversation back on track. I have found that when I can sincerely say, "I'm puzzled by your response. What's happening for you?" I'm often able to redirect the conversation to the present circumstance.

Sometimes it is tempting to think we know what is behind someone else's behavior. For example, one manager reported after a year-end review that her employee was angry and resistant during the performance discussion.

"What did you see or hear that lead you to that conclusion?" I asked.

"After I told him that our senior manager wasn't sure what he did, he asked for examples," the manager said.

"That seems like a reasonable

Coming Events

January 20-22
Institute for Defense and Government Advancement Network Centric Warfare
Arlington, VA
www.idga.org

January 26-28
3rd Annual Conference on the Acquisition of Software-Intensive Systems
Arlington, VA
www.sei.cmu.edu/products/events/acquisition

February 1-4
3rd International Conference on COTS-Based Software Systems
Redondo Beach, CA
www.iccbss.org/2004

February 3-5
WEST 2004
Western Conference and Exposition
San Diego, CA
www.west2004.org

March 1-3
17th Conference on Software Engineering Education and Training (CSEET 2004)
Norfolk, VA
www.cs.virginia.edu/cseet04

March 8-11
Software Engineering Process Group Conference 2004
Orlando, FL
www.sei.cmu.edu/sepg

March 30-31
3rd Annual Southeastern Software Engineering Conference
Huntsville, AL
www.ndia-tvc.org/SESEC

April 19-22
2004 Software Technology Conference
Salt Lake City, UT
www.stc-online.org

request," I said.

"Well, when I started to tell him that no one should have to ask what he did, he got real quiet," she continued. "He was so angry he couldn't speak."

"Did you check that out?" I asked.

"Well, no. I could just tell!"

Of course, we cannot just tell. This manager's employee might have been mystified, hurt, ashamed, or angry. He may have realized that he was not going to receive any useful information from his manager and decided that silence was his best course of action.

Sometimes we see external signs that lead us to believe that someone is feeling a certain way, but unless we check it out, we just do not know.

Dealing With Strong Reactions

Sometimes people react strongly to feedback. What do you do if someone starts to cry when you are giving feedback? Avoid the temptation to rush to comfort. Nudge a box of tissues across the desk and remain seated and quiet. Stare at the floor if you must. When the crying stops, continue. If the crying continues for more than several minutes, ask the employee if he or she needs a few minutes to regain composure. Ask the employee to return in five to 10 minutes.

I had an employee storm out of my office and slam the door during a feedback conversation about appropriate behavior during technical design meetings. I rescheduled the meeting and when he arrived for the next feedback discussion, I started by saying, "In our last meeting, you chose to walk out. This is a scheduled performance discussion, and if you refuse to participate, I'll start the formal process with HR." He chose to stay. Some organizations classify refusing to have a conversation with your manager as insubordination, and consider it grounds for dismissal. Check with your HR department, company lawyer, or commanding officer.

If an employee starts to yell, tell him or her that you are interested in what he or she has to say, but that you cannot hear when he or she is yelling. If the yelling continues, ask the employee to stop yelling. If he or she does not stop, end the meeting. You can end the meeting by asking the employee to leave or by leaving yourself. If you feel physically threatened, call security. If the situation gets to this point, it is beyond day-to-day feedback. Get HR involved right away.

Providing feedback takes thought and effort. It can be intimidating to bring up issues with another person. When I find I am feeling it is a lot of work to bring up an issue, I ask myself this: If I were missing the mark on my work, or had a habit that was getting in the way of others doing their work, would I want to know? I always answer yes. Most people do.

As with many things in life, practice does make providing feedback easier. Start practicing with reinforcing feedback and minor course corrections in one-on-one meetings. Then when you need to bring up bigger issues, both you and your staff will have had a chance to learn about giving and receiving feedback.◆

References

1. Seashore, Charles N., Edith Whitfield Seashore, and G. M. Weinberg. What Did You Say? The Art of Giving and Receiving Feedback. Attleboro MA: Douglas Charles Press, 1992: 3-7.
2. Satir, Virginia. The New People Making. Mountain View, CA: Science and Behavior Books, 1988: 20-29.
3. Fournies, Ferdinand F. Coaching for Improved Work Performance. New York: McGraw-Hill, 2000: 117.
4. Buckingham, Marcus, and C. Coffman. First, Break All the Rules: What the World's Greatest Managers Do Differently. New York: Simon and Schuster, 1999: 48.

About the Author

Esther Derby is founder and president of Esther Derby Associates, Inc., a management consulting firm based in Minneapolis, Minn. Derby works with software project teams to start projects on a solid footing, assess the current state of the project, and capture lessons learned during and after the project. She also coaches and trains technical people who are making the transition into management. Derby is technical editor of STQE magazine and regular columnist for Stickyminds.com and Computerworld.com. Derby is publisher of the quarterly newsletter, *insights*. She has a Bachelor of Arts from the University of Minnesota and a Master of Arts in Organizational Leadership.

Esther Derby Associates, Inc.
3620 11th Ave. S.
Minneapolis, MN 55407
Phone: (612) 724-8114
E-mail: derby@estherderby.com

Successful Software Management: 14 Lessons Learned©

Johanna Rothman
Rothman Consulting Group, Inc.

Successful managers realize that they need to balance the needs of the business, the employees, and the work environment to be effective. In this article, the author summarizes her experiences in determining the work to accomplish and planning it, managing successful relationships with the group, and managing reactions to typical management mistakes.

Shortly after becoming a manager, I dragged myself home from work, flopped on the couch, and said to my husband, "This management stuff is hard. Nothing I learned in school prepared me for this people stuff. And that *management training*, that was just form-filling-out nonsense. The soft skills – dealing with people – are the hardest." My husband chuckled and commiserated.

If you are like me, and you started your professional career as a technical person, this *management stuff* is difficult to do. Not the forms, although the forms can be irritating, but the difficult part is knowing how to deal with people, and completing the work your organization expects of you. I have now had more than 15 years of management experience, and have learned a number of lessons about managing people.

Define the Manager's Role

When you become a manager, your role is to organize purposefully [1]. For me, that means creating an environment where people can perform their best work. As a software manager, that means I work to create business value by balancing the needs of the business, the employees, and the environment. There is no *one right way* to do this; every organization is different. However, the following lessons have served me well in numerous organizations.

1. Know What They Pay You to Do

I have been a manager of developers, testers, and support staff. You would think it would be easy to know what the company paid me to do. However, my mission as a test manager – to report on the state of the software – is sometimes different from what the organizations desire: to find the Big Bad Bugs before the customer does, or bless this software. Even my mission as a development manager – develop the team members as much as the software – was different from what another organization desired: create software just good enough that we can be bought out.

My mission does not have to be the same as yours, and you may modify your mission as your organization changes. However, delivering on your mission as a manager is what your organization pays you to do. What is important is to notice when your title, your mission, and what the company pays you to do are not synchronized.

One quality assurance (QA) manager said it this way, "My management only wants to me to manage the testing, not raise risks, look for process improvement opportunities, or even gather and report on what I think are standard metrics. My manager and I are both frustrated. Focusing on just the testing is wrong." This QA manager has at least one alternative – change his title so that he and the organization both know that he is not attempting to perform organization-wide process improvement, to clarify expectations in the organization.

> *"When you align yourself with your manager's priorities, you do the work they pay you to do."*

Doing what the organization pays you to do, and not doing what they do not pay you to do makes a huge difference in how successfully you and your group can accomplish your mission. Make sure you clarify your mission at your organization so you can create to-do and not-to-do lists. These lists help you plan the work – for you and your group.

One development manager who temporarily took over installations from the tech support people realized that he no longer had a development team, but an installation support team. The development manager put installations on his not-to-do list and developed a plan to move installations back to tech support.

When you align yourself with your manager's priorities, you do the work they pay you to do.

2. Plan the Work: Portfolio Management

It is easy to be reactive at work and feel buffeted by the requested changes of your group. It is harder and necessary to be proactive and plan your group's work, even if that work changes every week. For me, planning includes these activities: identifying the project portfolio (i.e., new work, ongoing work, periodic work, *ad hoc* work), developing strategies for managing the work for each project, and knowing what done means for each project. One of the questions I like to ask is, "How little can we do?" I do not want to shortchange any project, so by asking about the minimum requirements, I can accommodate more projects successfully.

Part of planning the work is assigning the people to projects. I assign people to one important project then allow them to take on little bits and pieces of less important work when they need a break or are stuck on the important project. I avoid context switching (moving from one unrelated task to another) as much as possible.

3. Accept Only One No. 1 Priority at a Time

I have worked for many managers who demanded that my staff and I work on several top-priority projects simultaneously.

Senior managers perform different work than first-line and middle managers. It is not possible for senior managers to work on more than one top-priority task at a time. However, because they tend to have more wait states in their work, these senior managers are under the illusion that they are working on several top-priority projects at the same time.

Middle and first-line managers can only work on one No. 1 priority task at a time. However, sometimes we confuse urgency and importance [2]. At one

organization, I would arrive at work in the morning, check my voice mail, and respond to all message requests. That took until noon. Again after lunch, I would check my e-mail and voice mail and run around responding to those urgent requests. After a week of this, I realized I was not performing any of the important work such as planning for the group and lab, reviewing critical development plans, or planning my hiring strategy. I also realized that although people marked their e-mails and voice mails *high priority*, they did not utilize the information I had given them at the time I responded.

I stopped responding immediately to urgent requests and re-planned my days. While I still checked voice mail and e-mail, I tended to ask more questions about the deadlines for requests. Prioritizing requests helped me manage my management time. I still had the problem of too many high priority projects coming into my group, so I asked my manager these questions:

- If you could have one project first, which one would it be?
- What are the consequences if we release any of these projects late?

We talked and negotiated which projects had to be completed when and why. When I understood the trade-offs between projects, I was able to manage the work coming into my group.

4. Commit to Projects After Checking With Your Staff

Business needs change. Sometimes your manager will grab you in the hall and say, "Hey, can you do this project now, and finish it in two months?" Or, a senior management planning committee will call you into its meeting and say, "We need this project now. Can you commit to it?"

It is very tempting to say yes. However, saying yes is exactly the wrong thing to do. You can say, "Let me check to see if my previous estimate is still accurate, and I'll get back to you before 5 p.m. today."

If you say yes, you are training your senior management to ask you for answers when you do not know the answers. You have also committed your staff to a project that may not be within the scope you originally estimated.

5. Hire the Best People for the Job

Especially if you manage many projects, your greatest leverage point is in hiring appropriate staff. Too often, we hire people who have similar technical skills and personalities as the people already in our groups. Hiring people who are *just like the ones we have now* does not always gain the best people for the job.

When you hire people your staff thinks are great, you increase morale in the group, and you increase your group's capacity over time. I recommend you develop a hiring strategy that identifies the technical and soft skills you are looking for, and that you choose a variety of techniques for interviewing.

I have found auditions [3, 4, 5] to be an essential technique for interviewing technical staff. I normally create 30- to 45-minute auditions to see how a person works in a particular setting. Auditions help candidates show what they can do. If you organize a congruent audition, you do not trip people up on esoteric ideas or jargon; you create a simplified situation that the candidate could encounter at work. Watching the candidate, or having the candidate explain their answers/results is a powerful interview technique.

"Hiring people who are just like the ones we have now does not always gain the best people for the job."

You can create auditions for any position, including project managers, developers, testers, writers, support staff, analysts, systems engineers, product managers, program managers, and people managers. Define the behaviors you require in a position, and then create an audition using your products or open source products to see the person at work. Create auditions that are 30 minutes long to start. If you are having trouble deciding between multiple candidates, define another audition that is one hour long and invite the candidates back to see how they manage that audition. Auditions show you how the person works at work – priceless information.

I also recommend behavior-description interview questions [5, 6] to understand how a candidate has performed in previous jobs. Behavior-description questions are open-ended and ask the candidate to tell you the story of previous work. For example, to understand how a project manager deals with a project team who has not yet met a schedule, ask this closed question: "Have you ever managed a project where the team had trouble meeting the schedule?" If the answer is no, you can decide if the project manager has enough experience to manage your team. If the answer is yes, ask the open-ended behavior-description question, "What did you do? What actions did you take on that project to help the project team meet the schedule?" The answers you hear will help you assess that candidate's ability to work in your organization.

6. Preserve Good Teams

Part of my hiring strategy is to hire people who fit into my already-existing team, but sometimes you inherit teams or a project has completed and a team is ready to move on. When a team is successful, I try to keep them together so they can continue working well together. I may bring more people into the team, one at a time, especially if the team has been highly productive. But I do not scatter the productive team and hope they will form more productive teams. That just reduces their productivity.

Teams can overcome bad management and bad processes, but they cannot overcome a team un-jeller. A team un-jeller is the person who walks into the lunchroom, and suddenly everyone else leaves. Or, the un-jeller creates an argument out of every conversation. If you have a team un-jeller, find another place for that person to work, preferably for your competitor.

7. Avoid Micromanaging or Inflicting Help

Many of us were software developers, testers, analysts, or some other technical role before we became managers. When we were technical contributors, we knew how to perform the technical jobs. However, once you have been a manager for a while, you probably will not know precisely how to perform the employee's job.

I once had a boss who liked to creep into my office, stare over my shoulder, and say, "On line 16, shouldn't that be a …" By the time he reached "16," I had jumped out of my chair, become flustered, and lost my concentration. Micromanagement neither gets the job done faster, nor does inflicting advice or help.

On the other hand, you and the employee both need to know that the employee is progressing. I ask my staff to decide when they have been stuck for too long (time-box the work). Some tasks

require weeks of study, but most tasks require days or hours. If the employee spends more than the agreed-upon time on the task, their job is to ask for help. As the manager, your job is to find them help, not necessarily inflict your help.

8. Treat People Individually and With Respect

Buckingham and Coffman [7] claim that each employee's relationship with his or her manager is key to that employee's success and long-term happiness in the organization. That means we need to treat people fairly, but uniquely, so that we build and maintain the best possible relationships with each employee.

Everyone has his or her own preferences, especially in their communication patterns, and how they organize their thoughts about work. Some people prefer e-mail communications; some prefer in-person discussions. Some people want to understand all the reasons behind your requests, and others will take the request at face value. Some people need to gather data to make decisions; others will develop a model about the situation and make a decision based on that model.

It does not matter if people work top-down or bottom-up, or if they want to talk in person or by e-mail. What matters is that you, within reason, accommodate everyone's uniqueness.

I once managed two very talented developers who shared a large office. Begrudgingly, they allowed me to have 20-minute one-on-ones with each of them every two weeks. In between, if I wanted to talk to either of them, I had to e-mail them first – dropping in was not allowed. I treated them differently than the other people in my group, but fairly, considering their preferences.

They frequently worked on the same software. They never spoke to each other aloud, they only communicated via e-mail even though they shared an office. Because they were so successful at their work together, and even mentored others in the organization by e-mail, their communications preferences were a bit odd but acceptable. If I had tried to change them to meet my needs and work with them the same way I worked with the other people, none of us would have been happy.

9. Meet Weekly With Each Person

Even if you have hired stars, you still need to know each person's progress on their tasks, and how the project as a whole is progressing. I use one-on-ones weekly to meet with each person. We discuss the employee's progress on his or her tasks. Sometimes, tasks are amorphous and it is difficult to know when to stop or if the employee needs help. I ask each employee to show me visible progress on each task: drafts of plans, multiple designs, prototype test results, anything that shows me the employee is making progress and is not stuck. If the employee needs help completing the task, we discuss what kinds of help are appropriate.

I receive many benefits from these weekly meetings. I learn what everyone is doing and can track it in my notebook. It is easy to write up useful performance evaluations, including examples of successful and not so successful actions the employee has taken over the year. And, because we meet weekly, I can give feedback then, not when we make time. I also reduce the number of staff interruptions because everyone knows they can ask me non-urgent questions during the one-on-one. I can perform weekly career development and learn if my staff has personal issues affecting their ability to do their jobs.

"Buckingham and Coffman [7] claim that each employee's relationship with his or her manager is key to that employee's success and long-term happiness in the organization."

If I am managing more than eight people, I meet biweekly with more senior staff because they need less direct supervision.

Some of you are probably thinking you do not have time to meet with everyone once a week. However, if you do not set up specific times to meet with everyone, you tend to either not know what people are doing, or you are interrupted frequently by your staff with questions.

10. Plan Training Time Each Week

Technical work is constantly changing; most of the technical people I know enjoy learning new things. If you have a budget for formal training, that is great. Even if you do not have a budget, plan weekly training time in the form of brown-bag lunches, presentations from other groups in your organization, an internal user-group meeting of one of your tools, or presentations from people in your group about their successes or difficulties.

I use the weekly group meeting as a time to deliver the training. When I managed development groups, I organized this internal training, including technical leads of other sub-projects to explain their architecture and application programming interface (API) to other groups, testers to explain patterns of defects they found, different techniques for peer review, or discussion of a particularly interesting article in one of the technical magazines someone had read.

11. Fire People Who Cannot Perform the Work

Even when you meet regularly with your staff and encourage them to acquire help when they need it, some people in your group may not be able to perform at the level you require. First, make sure you have been specific and have given feedback to the employee with examples of inadequate behavior. If the employee understands the lack of performance, you can choose whether to coach the person or perform a get-well plan, or in radical circumstances, escort the employee out the door.

Retaining non-productive employees has direct and indirect costs. The direct costs are easier to define: You are paying a salary and benefits and not receiving the expected work. The indirect costs are much subtler and more damaging.

When you continue employing an inadequate employee, the morale of the entire workgroup declines. If morale declines enough, your best people will leave. Not only do you have someone in your group who is not successful, that person has driven away the people who are the most successful.

In addition to low morale, you and your group accomplish less than you expected. You are not just accomplishing less because of the one employee who cannot work at the level you require; that person probably has to hand off work to others in the group, and those other people will be delayed by the inadequate work.

I once inherited a group where the previous management had *spared* an employee from layoffs because he was

having personal problems. Those personal problems affected his work – he did not always come to work, he was late on every deliverable, and he was unable to perform most of his work. In my one-on-ones with the employee, I gave him examples of his work and asked if he was able to continue to work. He said yes. (If he had said no, we would have put him on short-term or long-term disability.) We chose to perform a get-well plan, which the employee stopped after a week. After the employee left, the morale in the group jumped dramatically and we were able to accomplish more work.

12. Emphasize Results, Not Time

I have worked for senior managers who rewarded individuals based on their work hours, i.e., those who started early and stayed late. Unfortunately, these managers had no ability to understand the results the long-working employees imposed on the rest of the organization: buggy code, inadequate designs, and tests that did not find obvious problems. When people work long hours, their productivity decreases, not increases [8]. In "Slack" [3], Tom DeMarco says, "Extended overtime is a productivity-reduction technique." The longer people stay at work, the less work they do. Instead, they perform the life activities they are not performing outside of work.

Make it possible for people to only work 40 hours a week. The less overtime people put in, the better their work will be.

If people tell you they are working long hours because they cannot accomplish anything in their regular work weeks, ask them where they spend their time. Look for patterns such as multi-tasking, or meetings that do not have any productive output. Use your management power to discover and remove the obstacles preventing people from working a 40-hour week.

13. Admit Your Mistakes

Sometimes, those obstacles to people completing their work successfully in 40 hours arise from your management mistakes. It is difficult, and sometimes embarrassing to have to admit you have made a mistake. In my experience, when I admitted mistakes to my staff, they have respected me more for it.

14. Recognize and Reward Good Work

Money is not an adequate reward for many technical people. If people think they are paid fairly, then more money is not reward enough. Recognition of good work and the opportunity to perform meaningful work [9] is much more important. Lack of money can be a demotivator, but only money is not sufficient when recognizing good work.

Kohn says, "[Rewards] motivate people to get rewards." If your organization has trained employees to expect money as a reward, this appreciation technique may seem small. Try it anyway.

When I use appreciation as a recognition technique I say, "I appreciate you, Jim, for your work on the blatz module and API definition. Your work made it possible for Joe to write great tests and for me to predict the project's progress." Appreciation between peers could mean even more than money from you. When you appreciate a person for good work and you explain what the work meant to you, you are motivating the person to continue performing similar work.

In addition, consider time off, group activities, movie tickets, or funny awards such as *best recursion of the week* as recognition techniques.

The most important part of a reward is to make sure it is congruent with each person's performance. Your staff knows who is performing well and who is coasting. If you recognize and reward evenly, you are not differentiating between outstanding performance and adequate performance. Make sure you reward a person's entire contribution (the entire work product, including how good the work product is, the timeliness of the deliverable, the person's ability to work with others, and whatever else is important to you), not just the size or quality of the work.

Summary

Managers exist to help people do their best work to serve the business of the organization. Technical people can make great managers as long as they understand people and want to succeed at working with them. Many successful technical managers took the time to learn about management, putting as much effort (if not more) than the effort they took to learn the necessary technical background for the technical jobs. Managers do not have to be perfect; they have to be good enough to create a working environment for their employees to deliver great work.◆

Acknowledgements

I thank Dwayne Phillips and the CROSSTALK reviewers for their input on this article.

References

1. Magretta, Joan. What Management Is: How It Works and Why It Is Everyone's Business. New York: The Free Press, 2002.
2. Covey, Stephen R. The Seven Habits of Highly Effective People. New York: Simon & Schuster, 1989.
3. DeMarco, Tom. Slack. New York: Broadway Books, 2001.
4. Weinberg, Gerald M. "Congruent Interviewing by Audition." Amplifying Your Effectiveness: Collected Essays. New York: Dorset House, 2000.
5. Rothman, Johanna. Hiring Technical People. New York: Dorset House, 2003.
6. Janz, Tom, et al. Behavior Description Interviewing. Englewood Cliffs, NJ: Prentice Hall, 1986.
7. Buckingham, Marcus, and Curt Coffman. First, Break All the Rules: What the World's Greatest Managers Do Differently. New York: Simon & Schuster, 1999.
8. DeMarco, Tom, and Tim Lister. Peopleware: Productive Projects and Teams. 2nd ed. New York: Dorset House, 1999.
9. Kohn, Alfie. Punished by Rewards. New York: Houghton-Mifflin, 1993.

About the Author

Johanna Rothman consults on managing high technology product development, which helps managers, teams, and organizations become more effective. Rothman uses pragmatic techniques for managing people, projects, and risks to create successful teams and projects. A frequent speaker and author on managing high technology product development, she has written numerous articles and is now a columnist for Software Development, Computerworld.com, and StickyMinds.com. Rothman served as the program chair for the Software Management conference and is the author of "Hiring Technical People."

Rothman Consulting Group, Inc.
38 Bonad Road
Arlington, MA 02476
Phone: (781) 641-4046
Fax: (781) 641-2764
E-mail: jr@jrothman.com

Deciding to Act

Walt Lipke
Oklahoma City Air Logistics Center

When should a manager act to correct a project that is not performing well? What should a manager do if he or she decides to act? How does a manager know that his or her action is sufficient? These are age-old questions. A poor outcome is a certainty if the manager's decision and action are not appropriate. This article discusses these questions and the manager's considerations. It concludes with a description of the Decision Logic diagram linking project performance with other factors to possible management actions.

As managers, we worry about delivering a quality product that performs as the customer expects. It is management's job to guide the project team to meet the negotiated commitment of technical performance, cost, and delivery date. It is tough to do.

There are innumerable opportunities to negatively impact the project throughout the entire performance period. Several critical elements such as personnel, facility, data, equipment, material, training, and subcontractors have the potential to overcome the best of plans. It is not difficult for anyone with project management experience to recall instances when each one of these elements caused additional cost and consumption of schedule.

To the best of the project team's ability, the risks associated with the critical elements are assessed. Subsequently, both cost and schedule reserve are created to mitigate the foreseen risks. Oftentimes however, to be competitive, project estimates and reserves are *squeezed*, thereby creating a poor situation for the manager from the outset – an aggressive plan with inadequate risk mitigation resources.

In the preceding paragraphs, I have stated the universal dilemma of project management, "Build me a Ferrari on a Yugo budget." Certainly, this is a gross overstatement but as a project manager, it is the way you feel. You understand, very well, from the first day that the probability of success is not 90 percent. It is more likely to be 60 percent, at best. Therefore, a small amount of inefficiency caused by risk impacts will nearly consume the project's reserves.

The execution of the project plan with no variation is the most efficient manner of performance. When changes are made to compensate for critical element impacts, inefficiency is created and some of the reserves are consumed. Therefore, to judiciously use the reserves, managers must have confidence that the change they induce will be beneficial; i.e., the project will have a greater opportunity to complete within the cost and schedule commitment.

The remainder of this article will create an approach for project analysis and decision making. The approach will address the following:

"It is common knowledge we should not react to insufficient data. However, sometimes the pressure to do something is overwhelming, and we act foolishly."

- When a manager should act.
- What action the manager should take.

A third aspect concerning the sufficiency of the action taken will also be discussed.

Project Management

Performance efficiency is measured by earned value management (EVM) indicators; i.e., the cost and schedule performance indexes, CPI and SPI, respectively[1]. Project managers using earned value in their management practice, thus, have a set of indicators that provide information concerning the health of their project. If the project is performing at the planned efficiencies (CPI and SPI equal to 1.0), the project is forecast to complete at the planned cost, and deliver its product on the expected delivery date. In addition, none of the planned cost or schedule reserves will be consumed.

One method to forecast whether or not a project will complete within its funding and negotiated delivery date is to compare the inverse indexes to ratios, which include the cost and schedule reserves[2]. When the value of CPI^{-1} is less than or equal to the cost ratio, the project manager has an expectation that the project will complete within the funding allocated. Correspondingly, if SPI^{-1} is less than the schedule ratio, the project is expected to complete by the negotiated completion date[3].

Of course, when the inverse indexes are greater than their respective ratios, the project manager knows his project is in trouble. The forecast indicates the plan will be exceeded, the reserves will be consumed, and more resources (time and funding) are needed. Understanding the project is failing, the project manager is inclined to take corrective action. Certainly the pressures from upper management and the customer compel the project manager to show that corrective action is already in progress.

Why is this the right thing to do? It may not be, but the project manager does not have anything in his tool kit to say he should do otherwise. Therefore, being *proactive* is his sole choice. Furthermore, the project manager knows that doing something, right or wrong, will buy time. Wishfully, within that time, a miracle happens and the project gets back on course. If good luck comes his way, the project is *righted*, and our hero receives a bonus and maybe even a promotion.

More than likely, the outcome of the corrective action taken will not be lucky. As mentioned previously, any change to the execution of the plan causes inefficiency. If the action taken is not the correct one, then management has inadvertently worsened the project performance and has not helped the situation. Subsequently, the manager, being proactive, takes another *shot in the dark*, likely worsening the situation once again. This process repeats until it becomes obvious to all concerned that the only way to deliver the product is to negotiate addi-

tional time and funding.

The outcome of this negative spiral is that the company and the project manager gain poor reputations. Additionally, if the product is extremely important and its sunk cost is significant with respect to the amount needed for completion, the agitated customer will likely agree to the added cost and delivery date extension. Under these circumstances, the company cannot expect repeat business with this customer.

Another common earned value approach is to manage using the cost variance (CV) percentage, i.e., CV divided by the planned cost (BAC). With this method, the project manager takes corrective action upon breaching an arbitrary limit, e.g., plus or minus 10 percent. It is common practice to ignore the schedule information and manage the project by cost variance alone. Generally, the results from the CV management method are as poor as described for the EVM indexes.

Certainly, there are successful projects that have been managed using earned value indicators; we are not implying EVM has no merit. Using earned value as a project management method greatly increases the opportunity for success, but improvement is needed. Project performance data is readily available, but rarely is it used advantageously. This is the state of today's management practice.

Analysis and Decision

Is there an alternative? Yes, there is. Simply reacting to poor performance indicators (CPI, SPI, or CV) is not good practice. There are other considerations needed to make the management decision. Including the aforementioned indicators of project performance, the manager needs information for the following areas:

- **Project Performance.** Do the indicators show poor project performance?
- **Sufficiency of Data.** Is enough data available to make a good decision?
- **Possible Strategy.** Can a strategy be created to recover the project?
- **Sufficient Time.** Is there enough time remaining to use the strategy?

By doing the analysis, and then answering these questions, a project manager can be confident the decision and action taken will have a much higher probability of success. Before moving on, a few words are needed concerning *Sufficiency of Data*. This information is critical in controlling management's tendency to overreact. It is common knowledge we should not react to insufficient data. However, sometimes the pressure to do something is overwhelming, and we act foolishly. Also, once a recovery strategy is implemented, we need to allow it time. It is not effective to amend and change strategies constantly; in fact, it is wasteful.

Supposing the questions can be answered, and a viable project recovery strategy can be prepared, *what actions are possible?* There are four basic actions:

- *No Action Required* when performance is good.
- *Investigate* when there is insufficient data.
- *Adjust/Realign* overtime or personnel.
- *Negotiate* cost, schedule, or requirements.

Connecting the analysis to the action is certainly not too difficult for the first two items. When the project is performing well, the manager would be wise to not make any changes. In addition, when the project has poor performance, but has insufficient data, it is prudent to investigate for potential causes and simply monitor the indicator(s) for improvement.

The Adjust/Realign and Negotiate actions are not so simply connected to the analysis results. The project manager should negotiate additional cost and/or schedule, or reduction of requirements, only when a recovery strategy is not possible or there is insufficient time for the recovery to be effective. Adjustment, i.e., raising or lowering overtime or number of project personnel, requires several inputs. It is the proper action when performance is poor, there is enough data to make an informed decision, a recovery strategy is possible, and there is sufficient time to execute it.

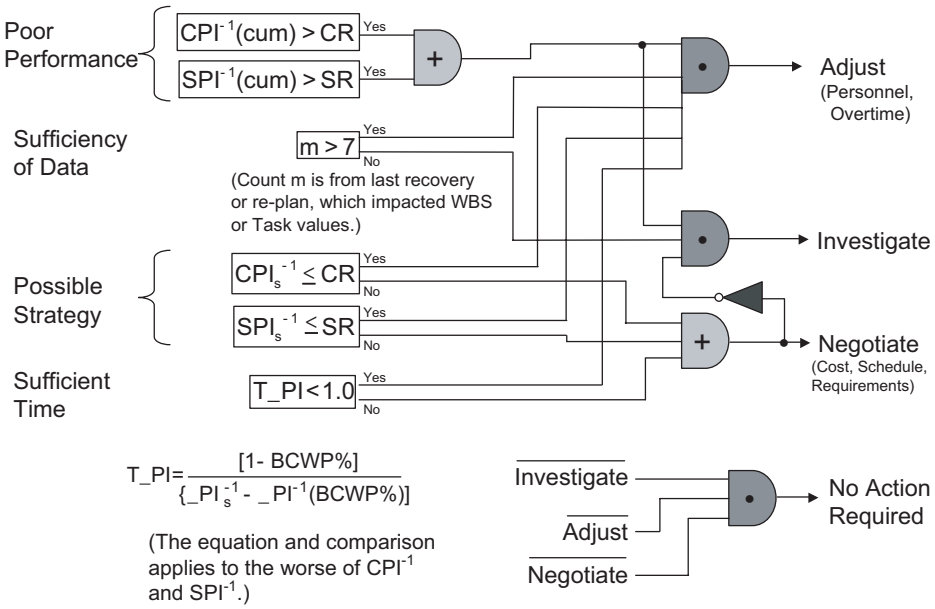
Careful realignment of personnel can yield increased efficiencies. However, the forecast effects of realignment cannot be quantified easily. It is recommended that this management action be used sparingly. Realignment can be an effective strategy when the values of CPI^{-1} and SPI^{-1} are less than their associated cost and schedule ratio, but worse than their planned value (1.0). Figure 1, Decision Logic, illustrates coupling the decision data to the management actions. The graphical diagram uses the logic symbols *and*, *or*, and *not* [4]. Once the inputs for *Poor Performance*, *Sufficiency of Data*, *Possible Strategy*, and *Sufficient Time* are known, the logic diagram can be used to identify the recommended management action.

When the cumulative value of either CPI^{-1} or SPI^{-1} is greater than its respective ratio, the project is performing poorly. Similarly, when there are more than seven periods of performance data, there is sufficient basis for taking action[5]. A possible strategy is one in which the forecast values of CPI^{-1} and SPI^{-1} at project completion are less than the cost and schedule ratios, respectively.

Developing a possible recovery strategy is a trade-off; improving one index negatively impacts the other [1]. For example, if the problem is poor cost performance, then the strategy, which causes its improvement, will detract from schedule performance, and vice versa. It is also to be noted that the project will experience an added expense to cost and schedule to implement the change.

Once the strategy has been determined, the To Complete Index (T_PI) is

Figure 1: *Decision Logic*



used to evaluate whether or not there is sufficient time for the recovery strategy to be successful[6]. When T_PI is less than 1.0, we are assured the strategy is viable. In other words, the project will not have to perform better than planned to achieve the customer commitments.

When the recommended action is either Adjust or Negotiate, management must then determine, *how much?* For Adjust, the project manager computes how many people to add or subtract from the project, or how much increase or decrease in overtime is needed to accomplish the recovery. For Negotiate, the manager determines the amount of overrun in cost and schedule. Knowing these values, he can then identify the requirements, which can be completed within the remaining time and funding, or the increases to schedule and cost needed to complete all of the requirements. Thus, the project manager has the data with which a contract change may be negotiated.

The calculation methods needed for Adjust, Negotiate, and Possible Strategy are beyond the scope of this paper. The reader may obtain the methods from [1].

Lastly, when Adjust, Investigate, and Negotiate are simultaneously inappropriate, the project requires no management action, i.e., No Action Required. The logic for this outcome is depicted in the lower right corner of Figure 1.

Example

To illustrate the use of the Decision Logic diagram in Figure 1, I will use hypothetical data. Let us suppose for this example the cost ratio (CR) equals 1.2, and the schedule ratio (SR) is 1.3. The reciprocals of the performance index values are 1.250 for CPI^{-1} and 1.125 for SPI^{-1}, respectively. The project is 40 percent complete (BCWP/BAC = 0.4) with 11 months of data.

If the project continues its present performance (CPI^{-1} exceeds CR), it cannot be completed within cost. However, the schedule performance provides some hope. Although schedule performance is not as good as planned, the project is expected to complete before the customer's delivery date (1.125 < 1.3). Therefore, a possible strategy is computed that elongates the schedule and improves cost efficiency. The possible strategy is determined to be SPI_s^{-1} and CPI_s^{-1} equal to 1.256 and 1.140, respectively. Using the CPI_s^{-1} strategy value (1.140), TCPI is computed to be 0.9375.

With all of the numerical information known, the logical comparisons can be made. We have a yes for Poor Performance; CPI^{-1} exceeds CR. Sufficiency of Data is yes; the value of m (11) is greater than seven. *Yeses* are evident for the Possible Strategy; both CPI_s^{-1} and SPI_s^{-1} are less than their respective ratios. In addition, Sufficient Time is yes; the computed value for TCPI is less than 1.0.

From the evaluation of the logical comparisons, the Decision Logic diagram is then used to identify the recommended management action. Investigate is not an appropriate management action because we have 11 months of data. We have also determined the recovery strategy is possible and there is sufficient time to execute it. Therefore, Negotiate is not the action to use. Adjust is the action the logic leads us to. Of course, with Adjust selected, No Action Required cannot be the recommended action.

For the Adjust action, the manager will perform calculations to determine either a revised overtime or staffing level. If all that is needed is a change in overtime, the success of the project recovery is more certain. Within reason, modifying the overtime level has much fewer repercussions than does changing staffing.

Summary

EVM provides incredible management information. However, it does not provide a good connection between the indicator values and the possible management actions. In today's project management climate, action is more likely to be taken because the project manager perceives it to be the correct thing to do in the eyes of the customer and his superiors.

The Decision Logic diagram provides the project manager with another tool. Using this tool, the method for deciding to act on a poorly performing project has been significantly refined. Furthermore, the action recommended is the one that will most benefit the project. The project manager now has a tool he or she can use effectively for managing his or her project, and for reporting his or her actions at the project reviews with both the customer and superiors. Using the decision diagram, the manager has supporting rationale for his or her actions.◆

References

1. Lipke, W. "Project Recovery … It Can Be Done." CROSSTALK Jan. 2002: 26-29.
2. Fleming, Q. Cost/Schedule Control Systems Criteria, The Management Guide to C/SCSC. Chicago: Probus, 1988.

Notes

1. The definitions of the cost and schedule performance indexes (CPI and SPI, respectively), and cost variance (CV) are:

   CPI = BCWP/ACWP
   SPI = BCWP/BCWS
   CV = BCWP —ACWP

   where,

   ACWP = Actual Cost for Work Performed
   BCWP = Budgeted Cost for Work Performed (earned value)
   BCWS = Budgeted Cost for Work Scheduled (project performance baseline)

   For more in-depth explanation of earned value and its indicators, see reference [2].
2. The definitions of the cost and schedule ratios are as follows:

   Cost Ratio = (BAC + MR)/BAC
   Schedule Ratio = (POP+SR)/POP

   where,

   BAC and MR are the EVM terms, Budget at Completion and Management Reserve (cost reserve), respectively. POP is the period of performance and SR is the schedule reserve, measured in units of time.
3. Although SPI, as defined by EVM, may be used, it is recommended to use the cumulative value of SPI(t). The time definition of the schedule performance index is:

   SPI(t) = ES/AT

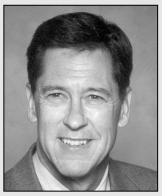
   where,

   AT is the actual period of time from project start to present, and ES is the resultant time associated with BCWS, when evaluated at the cost equivalent to the earned value (BCWP).
4. Reference Figure 1 for this discussion of the logic symbols. The *and* symbol is identified by the heavy dot. The operation of *and* is all of the inputs (lines from the left) must be *yes* for the output to be *yes*. The *or* symbol has the + sign. For the *or* operation, the output is *yes* if any of the inputs are *yes*. The *not* symbol is the triangle with a circle at its point. Its operation is to change the input (line from the right) from *yes* to an output of *no*, and vice versa.
5. The criteria for data sufficiency is that we must have, at minimum, 50 percent confidence of knowing the true values of the performance indexes, CPI_t and

SPI_t. More than seven periods of performance data are needed for the cumulative quantities of CPI and SPI to meet this requirement. Statistically, CPI_t and SPI_t are known to the degree that, at minimum, it is 50 percent probable that they are within plus or minus one-fourth of the standard deviation of the periodic index values from their respective cumulative values.

6. The equation for the To Complete Index (T_PI) is shown on Figure 1. The underline spaces in the symbols are to be filled in with either *S* or *C*, indicating schedule or cost, respectively. For example, when TSPI is calculated, *S* would be filled in for the other blanks in the equation's denominator. The symbol BCWP% represents BCWP divided by BAC.

About the Author



Walt Lipke is the deputy chief of the Software Division at the Oklahoma City Air Logistics Center. The division employs approximately 600 people, primarily electronics engineers. He has 30 years of experience in the development, maintenance, and management of software for automated testing of avionics. In 1993 with his guidance, the Test Program Set and Industrial Automation (TPS and IA) functions of the division became the first Air Force activity to achieve Level 2 of the Software Engineering Institute's Capability Maturity Model® (CMM®). In 1996, these functions became the first software activity in federal service to achieve CMM Level 4 distinction. The TPS and IA functions, under his direction, became ISO 9001/TickIT registered in 1998. These same functions were honored in 1999 with the Institute of Electrical and Electronics Engineers' Computer Society Award for Software Process Achievement. Lipke is a professional engineer with a master's degree in physics.

OC-ALC/MAS
Tinker AFB, OK 73145-3312
Phone: (405) 736-3341
Fax: (405) 736-3345
E-mail: walter.lipke@tinker.af.mil

Web Sites

Product Development and Management Association

www.pdma.org

The Product Development and Management Association's (PDMA) mission is to improve the effectiveness of people engaged in developing and managing new products – both new manufactured goods and new services. This mission includes facilitating the generation of new information, helping convert this information into knowledge that is in a usable format, and making this new knowledge broadly available to those who might benefit from it. A basic tenet of the PDMA is that enhanced product innovation represents a desirable and necessary economic goal for firms that wish to achieve and retain a profitable competitive advantage in the long term.

What You Need to Know About Management

www.management.about.com

The About Web sites are a network with each site run by a professional Guide who is carefully screened and trained by About. Guides build a comprehensive environment around each of their specific topics, including the best new content, relevant links, how-to's, forums, and answers to just about any question. What You Need to Know About Management includes general, people, and project management; leadership; communications business ethics; conflict resolution; and more. The Guide for this site is F. John Reh, who has a 30-year management career.

New Grange Center for Project Management

www.newgrange.org

The New Grange Center for Project Management is a nonprofit, all-volunteer organization dedicated to the principle of building a community of practice among project managers. Its goal is to get to the heart of project management by defining what really works and why. The backbone of the organization is its five-minute e-mail list where members take five minutes to address what they learned from their latest problem, which over time develops into a database. Topics include how to develop a project communication plan, how to define the best reward structure, and the right way to conduct a project post-mortem review.

Integrated Software Industry Benchmarking Association

www.isiba.com

The Integrated Software Industry Benchmarking Association (ISIBA) is a free association of software companies. ISIBA conducts benchmarking studies to compare operating performance and identify practices that improve the overall operations of its members. Consortium studies are offered to the membership as a whole with costs divided. Single-company sponsored studies addressing the interest of one member company can be offered to other selected members for no fee. Interest group roundtables are organized throughout the year.

Project Management Institute

www.pmi.org

Established in 1969, the Project Management Institute (PMI) is a not-for-profit, project-management professional association with over 100,000 members in 125 countries. PMI members are in many different industry areas, including aerospace, automotive, business management, construction, engineering, financial services, information technology, pharmaceuticals, and telecommunications. PMI publishes "A Guide to the Project Management Body of Knowledge," and its Project Management Professional certification is the world's most recognized professional credential for individuals associated with project management. In 1999, PMI became the first organization in the world to have its Certification Program attain International Organization for Standardization 9001 recognition.

Software Program Managers Network

www.spmn.com

The Software Program Managers Network (SPMN) is sponsored by the deputy under secretary of defense for Science and Technology, Software Intensive Systems Directorate. It seeks out proven industry and government software best practices and conveys them to managers of large-scale Department of Defense software-intensive acquisition programs. The SPMN provides consulting, on-site program assessments, project risk assessments, software tools, guidebooks, and specialized hands-on training.

Requirements Engineering Maturity in the CMMI

Dennis Linscomb
Computer Sciences Corp.

Much has been written on requirements engineering (RE) but very little about RE maturity. Is there such a thing? If so, why and how do you measure it? This article discusses these topics and analyzes how the Capability Maturity Model® Integration addresses RE maturity.

The discipline relating to the systematic handling of requirements has typically been called requirements engineering (RE) [1]. One definition of RE that is regularly cited in RE literature is,

> … the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families. [2]

Many articles and books have been written on the components of RE and their interrelationship [3]. In the Institute of Electrical and Electronics Engineers' (IEEE) Software Engineering Body of Knowledge (SWEBOK) [4], the Software Requirements Knowledge Area consists of the following components: RE Process, Requirements Elicitation, Analysis, Specification, Validation, and Management. These components are common to the RE literature. While a lot has been written about RE scope, components, techniques, templates, and tools, there has not been a lot written about RE maturity[1]. Is there such a thing as progressing in RE from a basic to an advanced level? If so, how do you define it, and why should you measure it?

In my information technology (IT) experience in the software applications area at several companies, I have clearly seen levels of RE maturity. I think you will agree with me when you consider the following scenarios in Table 1.

Of course, many more scenarios could be listed, but I think I have made my point. The harder questions to answer are these: (1) Why do you need to define levels of RE maturity? (2) Assuming there is a good reason to define levels, how do you define them, i.e., what criteria do you use?

The question "why define RE maturity" is usually part of a larger question: Why define and measure the process maturity (usually software process maturity) of an organization? The main reason many organizations do it (at least from the executive management viewpoint) is economic, namely they want to get more business and retain existing business. One would hope that they also (primarily?) do it because it is the right thing to do, but that is not always the main motivator. The Software Engineering Institute's (SEI) Capability Maturity Model® (CMM®) Integration (CMMI®)[2] usually comes to mind when discussing a process maturity rating. Using this model, organizations are given a rating of Maturity Level (ML) 2-5[3] via formal assessments.

If you are not in an organization striving for a certain ML, or if the strategy of your organization is something other than operational excellence[4], then much of the rest of this article is not meant for you. If your organization fits this profile, I recommend pursuing RE in a way that makes sense for your organization's goals and strategy. However, assuming you can find a reason for rating the process maturity of your organization, then it is appropriate to analyze how best to fit RE maturity into your process model.

For my analysis of RE maturity, I chose the CMMI not only because it is widely used but also because it is one of the few[5] process models that attempts to define levels of maturity for IT-related processes. The CMMI defines two process areas (PAs) relating to RE: Requirements Management (REQM) and Requirements Development (RD). Although RE affects many more CMMI PAs due to its importance in the software development life cycle, these two PAs are the ones in which RE is specifically addressed. The REQM and RD PAs are measured for their maturity based on the type of CMMI representation of the model you are using.

In the CMMI Staged Representation, all PAs are defined at one of four MLs (2-5, with 5 being the most mature). This representation puts REQM at ML2 and RD at ML3. As you mature through the MLs, you must continue to perform at the previous MLs. Therefore, to implement RD means that you have institutionalized[6] REQM.

In the CMMI Continuous Representation, one of six capability levels (0-5, with 5 being the highest capability) is assigned to *each* PA. Theoretically (though not practically), an organization could be at a high capability level (e.g., 5) for REQM and a low capability level (e.g., 0) for RD.

Therefore, no matter which representation you use, the CMMI model describes a progression from less RE maturity to more RE maturity. At ML3 (in the Staged Representation) or capability level 3 (in the Continuous Representation), an organization is considered to be more mature[7] in RE than they would be at previous levels.

Although the CMMI is now being widely used and is at version 1.1, I think it still makes sense to ask the question, "Does the CMMI currently define RE maturity the way it should be defined based on industry standards and practice?" My answer is, "No," based on RE terminology and on the typical order of RE activity.

Table 1: *Requirements Engineering Maturity Levels*

Less Mature in RE	More Mature in RE
Requirements are taken verbally over the telephone from one stakeholder.	Requirements are documented after getting consensus from multiple stakeholders.
Only one requirements elicitation/gathering technique is used without regard to the nature of the stakeholders or the project.	Several requirements elicitation/gathering techniques are known and used based on the type of project and the mix of stakeholders.
The original requirements are documented in a repository but are never modified as individual requirements change over time.	A repository of up-to-date user-approved requirements is maintained throughout the life of the project.
There is no change control process defined for requirements or, if defined, it is never consistently used.	A requirements change control process is defined and consistently used.
There is no way of knowing whether or not every requirement was implemented.	A requirements traceability matrix is developed and maintained.

With respect to terminology, it should be noted that CMMI treats the standard RE components (management, elicitation, analysis, specification, and validation) differently from that usually found in RE literature. For example, REQM is defined as a separate PA, but requirements elicitation, analysis, specification, and validation are all lumped into one RD PA. I have not found any SEI documentation describing the rationale of their taxonomy, as does the SWEBOK [5]. Part of the answer may lie in the fact that the RD PA in the CMMI was split out of the Software Product Engineering PA in the CMM. This difference in terminology is more than academic. By placing REQM and RD not only in separate PAs but also in separate MLs, there is an artificial dichotomy created between the components of RE. As I shall discuss later, REQM cannot be done in a vacuum.

At this point, you may object that I am mixing apples and oranges. Requirements management, elicitation, analysis, specification, and validation are categories or a taxonomy of RE activities, one may argue, whereas the CMMI is concerned with describing process areas relating to RE. However, these categories may also be viewed as activities in the RE process. According to Linda Macaulay,

> In general terms, the RE process can be thought of as a series of activities consisting of articulating the initial concept, problem analysis, feasibility and choice of options, analysis and modelling [sic], and requirements documentation. [6]

Requirements life cycles have been defined as consisting of three to five phases with the above RE categories, or equivalent terms, as phase names[8]. Although the CMMI does not require you to choose any specific RE life cycle, it should use standard RE terminology in describing PAs, goals, and practices relating to RE.

With respect to the typical order of RE activity, I believe there is room for improvement in the CMMI. While the CMMI does not dictate any specific RE life cycle, it does have something to say about the order of implementation and institutionalization of RE by its placement of a certain RE activity under a specific ML. I contend that this order is not always logical. Consider the following examples:

1. Requirements elicitation is supposed to be institutionalized in the ML3 RD PA under Specific Goal (SG) 1. However, under the ML2 REQM PA, you are supposed to be managing these requirements. How can you manage them at ML2 if you do not have an institutionalized way of eliciting requirements until ML3? The ML2 REQM Specific Practice (SP) 1.1 "Obtain an Understanding of Requirements" does not contain enough detail about the scope, source, and specificity of requirements to form a solid basis for managing those requirements at ML2. Requirements-related problems are closely tied to project failure[9]. Why wait until ML3 to institutionalize practices to ensure that you have complete and accurate requirements?
2. Requirements analysis and validation are defined under the ML3 RD PA (under SG 3). However, you need to do a certain amount of analysis and validation of requirements at ML2 in order to get them in a mature enough state to manage them.
3. Bidirectional requirements traceability is required under the ML2 REQM PA. While a certain amount of requirements traceability is necessary at ML2, should an organization concentrate on this full-blown bidirectional traceability before institutionalizing requirements elicitation and analysis (at ML3)? I think not. It is interesting to note that Rational Software puts traceability at Level 4 in their Five Levels of Requirements Management Maturity [7].

The CMMI recognizes that there is RE activity present even in ML1 organizations[10]. Also, the CMMI acknowledges the interrelationship of RE activities in the Introductory Notes to the REQM PA:

> … if the Requirements Development process area is implemented, its processes will generate product and product-component requirements that will also be managed by the requirements management processes. When the Requirements Management, Requirements Development, and Technical Solution process areas are all implemented, their associated processes may be closely tied and be performed concurrently. [8]

Therefore, the issue is not that the CMMI is opposed in principle to a normal progression and maturity of RE activity. The issue is whether the CMMI defines it the best way, i.e., using terms and maturity criteria that the industry can agree upon, and puts RE at the appropriate maturity levels.

The following is my proposal for the SEI CMMI Project Team:

1. Review the entire RE discipline (and not just the requirements-related goals and practices currently in the CMMI) with the goal of determining how RE should be presented in the CMMI. The review should include holding CMMI workshops to get consensus from a broad spectrum of RE practitioners about what they consider to be basic versus advanced requirements practices.
2. Work closely with the IEEE to ensure that their standards and work products, e.g., SWEBOK and the CMMI stay in sync with respect to terminology and processes.
3. Revise the CMMI model to reflect consensus from the above steps.

I think consensus from this effort will support the following concepts:

1. RE maturity should be represented at more than one ML. It is just not practical to assume that an organization can and should implement everything related to RE at one level.
2. A RE-related PA should, at minimum, exist at ML2 and ML3. Perhaps a case can be made for some advanced RE activity at ML4 and ML5. However, until that case is made, I believe the CMM and CMMI are correct in placing RE activity at ML2 and ML3.
3. The dichotomy between requirements management and other RE activities should be minimized.

Based on my IT experience, my recommendation (though I am willing to change it based on consensus from the above proposal) is that the CMMI Staged Representation should be changed to something like a Basic RE PA at ML2 and an Advanced RE PA at ML3. The concept of basic and advanced is not foreign to the CMMI. For example, there are basic and advanced project and process management PAs [9].

The following are my recommendations for some of the goals and practices at Basic RE PA (for ML2):

1. Elicit/gather requirements. You do not have to have a trained staff of facilitators and many different ways of eliciting or gathering requirements at ML2. You just need at least one repeatable method of obtaining project requirements. Why wait until ML3 to institutionalize one method?
2. Analyze requirements. To ensure they meet the characteristics of good requirements, e.g., complete, clear, consistent, verifiable, traceable, feasi-

ble, and design independent. These characteristics are currently defined as examples in the ML2 REQM PA under SP 1.1. However, why use the ambiguous title "Obtain an Understanding of Requirements" when many ML 1 and 2 organizations know what you mean by requirements elicitation and analysis?

3. Document requirements. This is already in the ML2 REQM PA as a typical work product (an agreed-to set of requirements) under SP 1.1.
4. Get approval of requirements from appropriate stakeholders. This is already in the ML2 REQM PA as SP 1.2 (Obtain Commitment to Requirements).
5. Manage requirements changes. This is already in the ML2 REQM PA as SP 1.3.
6. Develop and maintain requirements traceability to the extent that you can demonstrate that all requirements have been implemented. See comment below on traceability at ML3.

The following are my recommendations for some of the goals and practices at Advanced RE PA (for ML3):

1. Develop different techniques of eliciting requirements, define criteria about when to use each based on project profiles, and institutionalize these techniques with formal training and mentoring.
2. Provide a staff (more than one – even if part-time) of trained requirements facilitators.
3. Develop and maintain a full-blown, bidirectional requirements traceability matrix showing that each requirement is satisfied in design, development, test, and implemented work products. I have never seen a ML2 organization do a good job at this type of full-blown traceability matrix. Yet it is required in SP 1.4 of the ML2 REQM PA.
4. Include all current ML3 RD goals and practices that involve showing interrelationship of requirements, requirements decomposition, assumed system requirements, and requirements change metrics. In other words, everything beyond the ML2 basics defined above.

Probably, some people may not want to tamper with REQM at ML2. They believe this PA simply follows the overall CMM process improvement road map to get management infrastructure in place at ML2 in order to support the engineering processes at higher levels[11]. They make the point that engineering processes are in effect at ML2, but they do not have to be documented and can be informal. While I agree with the CMM improvement strategy, it should not be interpreted in such a way as to exclude activities required to make work products mature enough to be managed at ML2.

In other words, you cannot manage in a vacuum. A certain level of formalization must be in place for some engineering practices in order for the management process areas to work properly. Consider the ML2 Project Planning PA. You need to perform a certain amount of technical (engineering?) activities for SP 1.4 (perhaps using some sophisticated tools) in order to get sound estimates of effort and cost so that you can put together the project plan in order to manage it. In like manner, the ML2 REQM PA assumes a certain amount of RE formalization and institutionalization in order to ensure that requirements are mature enough to be managed[12].

Also, it should be noted that ML3 has never been composed of pure engineering PAs. For example, management activities permeate the Integrated Software Management and Intergroup Coordination PAs in the CMM and several PAs in the CMMI, such as Integrated Project Management, Risk Management, Integrated Teaming, Integrated Supplier Management, and Decision Analysis and Resolution. That is the way it should be. Each ML should be composed of the correct mixture of technical and management activities so that management can be effective for that ML.

You may be asking, "If this RE maturity discrepancy is that obvious in the CMM/CMMI, why has it not been a problem for organizations that have attained ML2 or ML3?" My answer is twofold:

1. Some ML1 organizations fund their process improvement effort with the goal of achieving ML3. In other words, they are not first assessed at ML2 and then work toward ML3. Why? Because two separate efforts are more expensive than one. Also, they may be under management pressure to achieve ML3 by a certain date, and there is not enough time to do this in two independent efforts. Whatever the reason, by including both ML2 and ML3 in one process improvement effort, all of RE goals and practices are covered. Therefore, it never becomes an issue about how RE is split out between ML2 and ML3.
2. For those ML1 companies who work toward ML2 as their goal, they just know from past experiences and industry best practices that certain ML3 RE practices (e.g., elicitation and analysis) must be done as part of their life cycle. Therefore, they continue to do them because they make sense and are required to deliver quality work products.

In conclusion, I believe that RE maturity makes sense as a concept and reflects reality in IT organizations seeking operational excellence, whether or not they call it basic versus advanced RE. The attempt of the CMMI to define this RE maturity is admirable but deficient. However, this deficiency does not mean that we abandon the model. The CMMI is being widely used, and I have personally witnessed the success of CMM at several companies. I want the model to continue its success. However, for it to be durable for many years to come, I believe it needs an overhaul in the RE area.◆

References

1. Abran, Alain, and James W. Moore. Guide to the Software Engineering Body of Knowledge – SWEBOK. Eds. Pierre Bourque and Robert Dupuis. New York: Institute of Electrical and Electronics Engineers, May 2001: 9 <www.swebok.org>.
2. Zave, Pamela. "Classification of Research Efforts in Requirements Engineering." ACM Computing Surveys 29.4 (Dec. 1997): 315-321.
3. Davis, Alan M. "Requirements Bibliography." <http://web.uccs.edu/adavis/UCCS/reqbib.htm>.
4. Abran 15ff.
5. Abran 23f.
6. Macaulay, Linda. Requirements for Requirements Engineering Techniques. Proc. of the Second International Conference on Requirements Engineering. York, United Kingdom, 1995. New York: IEEE Computer Society Press, Apr. 1996: 158.
7. Heumann, Jim. "The Five Levels of Requirements Management Maturity." The Rational Edge Feb. 2003 <www.therationaledge.com/content/feb_03/f_managementMaturity_jh.jsp>.
8. CMMI Product Team. CMMI Ver. 1.1. Pittsburgh, PA: Software Engineering Institute, Mar. 2002: 82.
9. CMMI Product Team Chapter 5.

Notes

1. See [7] for Rational Software's Five Levels of Requirements Management Maturity. Some articles describe a Requirements Maturity Index (RMI), but this has to do with the readiness of

requirements for design and development and not the maturity of the RE *process*. For an article that discusses RMI, see Stuart Anderson and Massimo Felici, "Quantitative Aspects of Requirements Evolution," <www.dcs.ed.ac.uk/home/mas/doc/camerarady_compsac2002.pdf>.

2. The original version of this model, called the Capability Maturity Model (CMM), is still in use. However, since the CMMI will eventually replace the CMM, most of my references are to the CMMI [8].
3. There is a Level 1 but this is a starting point for all organizations and does not represent a level of assessed maturity. Also, Levels 2-5 are based on the Staged Representation of the CMMI.
4. Stan Rifkin has written several articles on applying the main thesis of the book "The Discipline of Market Leaders" by Michael Treacy and Fred Wiersema to using the CMM and other process improvement efforts <www.master-systems.com/Papers.ivnu>.
5. The only other models I know about that define maturity levels for IT-related processes are in the CMM family (e.g., the Capability Maturity Model for Software Acquisition and the FAA integrated Capability Maturity Model) and Electronic Industries Alliance 731. If you know of other models, please e-mail me.
6. The CMMI defines *institutionalization* as "… the ingrained way of doing business that an organization follows routinely as part of its corporate culture" (see [8], Glossary: 579).
7. Although some proponents of the CMMI Continuous Representation say that a capability level is not a ML, I contend that it is in a certain sense of the word *maturity*. The CMMI defines a capability level as applying to an organization's process-improvement achievement for a certain process area. Therefore, as you progress in capability levels for a certain process area, are you not becoming more mature in that process area?
8. For examples of three and five phases, see Jawed Siddiqi and M. Chandra Shekaran, "Requirements Engineering: The Emerging Wisdom." IEEE Software Mar. 1996: 15-19. For an example of four phases, see Ian Sommerville, Software Engineering. Harlow, England: Addison-Wesley, 1996: 67f.
9. Numerous studies show that requirements play a large role in the success or failure of projects. The following are only a few: Standish Group's "Chaos Report" for 1994, 1997, and 2000. Karl Wiegers, Software Requirements. Microsoft, 1999: 5, 24.
10. "Certainly, we would expect maturity level 1 organizations to perform requirements analysis, design, integration, and verification. However, these activities are not described until maturity level 3 …" (see [8], Chap. 2 Model Components: 16).
11. CMM for Software Ver. 1.1, Section 2.2.2, p.15f, "Understanding the Repeatable and Defined Levels" states: "Level 2 provides the foundation for Level 3 because the focus is on management acting to improve its processes before tackling technical and organizational issues at Level 3. … Level 3 builds on this project management foundation by defining, integrating, and documenting the entire software process."
12. For examples of what happens if you try to do requirements management without requirements engineering and vice versa, see Nancy R. Mead's article, "Requirements Management and Requirements Engineering: You Can't Have One Without the Other." Cutter IT Journal May 2000.

About the Author

Dennis Linscomb is an employee of Computer Sciences Corp. (CSC) through CSC's acquisition of DynCorp. At DynCorp, he served as the quality assurance manager for the corporate Information Technology department. He has been in information technology for 28 years and has worked in several areas of applications software, including programming, analysis, testing, quality assurance, production support, and management. He has been involved in software process improvement and the Capability Maturity Model®/Capability Maturity Model Integration for about 10 years. He has a master's degree in business administration from Pepperdine University.

Computer Sciences Corp.
11710 Plaza America Drive
Reston, VA 20190
E-mail: dennis_linscomb@msn.com

CROSSTALK
The Journal of Defense Software Engineering

ARTICLE INDEX VOLUME 16

TOPIC	ARTICLE TITLE	AUTHOR(S)	ISSUE	PAGE
Process Improvement	Destroying Communication and Control in Software Development	Dr. G. M. Weinberg	4	4
	Experiences Applying the People Capability Maturity Model	Dr. B. Curtis, Dr. W. E. Hefley, S. A. Miller	4	9
	Highpoints From the Amplifying Your Effectiveness Conference	E. Starrett	2	27
	Life Cycle of a Silver Bullet	S. A. Sheard	7	28
	Obedience Training for Managers	V. Slavin, P. Kimmerly	4	14
Programming Languages	Evolutionary Trends of Programming Languages	Lt. Col. T. M. Schorsch, Dr. D. A. Cook	2	4
	Language Considerations	D. Ludwig	2	10
	SEPR and Programming Language Selection	R. Riehle	2	13
Project Management	Monitoring Progress in Software Development	J. van der Linden	7	31
	Overview of Project Management	T. Perkins, R. Peterson, L. Smith	1	4
	Planning and Managing the Development of Complex Software Systems	Dr. R. Bechtold	5	23
	The Probability of Success	W. Lipke	11	30
	Project Expectations: The Boundaries for Agile Development	D. Mekelburg	4	28
Quality	Comparing Lean Six Sigma to the Capability Maturity Model	Dr. K. D. Shere	9	9
	Delivering Quality Products That Meet Customer Expectations	L. S. Wheatcraft	1	11
	Lean Six Sigma: How Does It Affect the Government?	Dr. K. D. Shere	3	8
Requirements	An Enterprise Modeling Framework for Complex Software Systems	Dr. P. Donzelli	2	23
	Requirements Engineering Maturity in the CMMI	D. Linscomb	12	25
Risk Management	Risk Management Applied to the Reengineering of a Weapon System	C. Y. Laporte, G. Boucher	1	24
Software Development	Application of Lightweight Formal Methods in Requirements Engineering	V. George, Dr. R. Vaughn	1	30
	Clarify the Mission: A Necessary Addition to the Joint Technical Architecture	I. Ögren	3	25
	International Standardization in Software and Systems Engineering	F. Coallier	2	18
	A Pair Programming Experience	Dr. R. W. Jensen	3	22
	Software Architecture as a Combination of Patterns	K. Petersson, T. Persson, Dr. B. I. Sanden	10	25
	Wireless Data Entry Device for Forward Observers	P. Manz, Lt. Col. J. R. Surdu, 2nd Lt. A. M. Adas, 2nd Lt. Z. R. Miller, 2nd Lt. A. J. Peplinski, 2nd Lt. E. J. Watson	7	31
Software Inspections	Determining Return on Investment Using Software Inspections	D. O'Neill	3	16
	High Quality, Low Cost Software Inspections	L. A. Poulin	1	29
Systems Engineering	Developing a Stable Architecture to Interface Aircraft to Commercial PC's	D. W. Christenson, L. Silver	11	26
Testing	Interface-Driven, Model-Based Test Automation	Dr. M. R. Blackburn, R. D. Busser, A. M. Nauman	5	27
	Let's Play 20 Questions: Tell Me About Your Organization's Quality Assurance and Testing	G. E. Mogyorodi	3	30
	New Spreadsheet Tool Helps Determine Minimal Set of Test Parameter Combinations	G. T. Daich	8	26
	What Is Requirements-Based Testing?	G. E. Mogyorodi	3	12
Top 5	2003 U.S. Government's Top 5 Quality Software Projects	M. Schaefer	9	4
	CrossTalk Honors the 2002 Top 5 Quality Software Projects Finalists	P. Bowers	7	16
	Defense Civilian Pay System Streamlines Payroll System Operations	C. Fortier-Lozancich	7	6
	The JHMCS Operational Flight Program Is Usable on Three Tactical Aircraft	C. Fortier-Lozancich	7	10
	Kwajalein Modernization and Remoting Project Replaces Four Unique Radar Systems With One Common Design	P. Bowers	7	12
	The OneSAF Testbed Baseline SAF Puts Added Simulation Capabilities Into Users' Hands	P. Bowers	7	14
	Software Project Winners Exemplify Software Development Best Practices	E. Starrett	7	4
	Tactical Data Radio System Enhances Combat Effectiveness	P. Bowers	7	8

CONTINUED ON NEXT PAGE

The CrossTalk staff would like to wish you and yours the very best this holiday season and the happiest of New Years.

Misbehaving Toys

It's toy time folks, and as the song goes, "You'd better watch out!" The bleeding thumbs and *tons of assembly required* so popular on past Christmas eves have been updated, so here's fair warning.

You might say I have issues regarding my kids' toys, and it all started in the middle of the night long after one Christmas revelry. I awoke hearing voices in the dead of night, a conversation going on in my living room. Naturally, I panicked and was soon wide-awake. My intruder turned out to be Cookie Monster, and he was being rudely interrupted by Elmo.

Cookie would say, "Cookie Monster here, Cookie see you." Somewhere in there, Elmo's squeaky voice would cut him off with several rounds of "Let's play." The fight would go on for a while, then stop, and then start up again hours later. I would walk into the living room and from the direction of the toy basket hear a gravelly, "Hey, scram," followed by Oscar's scariest laugh.

While I admit this feature could actually prove useful for clearing the stoop, we could never count on it, except to speak up when it felt like it. I last heard from the gang as I passed a sack in the garage bound for the local Goodwill, destined for another soon-to-be-sleepless house.

It's funny how these talking toys seem to go off at just the right or wrong moments. Consider this recent report: an acquaintance of my wife bought a talking one-eyed Mike of "Monsters Inc." fame. You no doubt remember the character from the movie; he's essentially a green ball with one large eye and sounds a lot like Billy Crystal. Anyway, Mike was stashed in the master closet, secreted away for Christmas morning.

Fates crossed when Grandma came to visit and at one point excused herself to the bedroom to change. At some point of undress she heard a strangely familiar man's voice in the closet greeting her with, "I've got my eye on you."

We understand Grandma did not require medical attention – even after hurdling a queen-sized bed – but her opinion of Billy Crystal did go down a notch. I suspect Mike has already made a pass through the Goodwill cycle.

Of course, many of the toys under the tree these days not only talk, but they also listen and respond to our commands – rather like real pets. I'm very impressed with two such creatures my girls have on their shelves; they respond exactly like our Schnauzer – which is to ignore all of my commands. Oh, I have managed to get them to beep and jump around a bit and flash their LED eyes, which shouldn't surprise me, since the Schnauzer raises a ruckus and jumps like crazy without any commands from me.

I could go on. There was the dollhouse that came with several realistic sounds for your typical household, including a crying baby that, yes, went off in the middle of the night. Then there was that cool toy bank that did all sorts of things, even more than was advertised on the box. I came downstairs one morning to find my young daughter asleep on the couch. I found the bank, buried in blankets, beeping non-stop in her room.

What manufactures don't do, the enterprising engineer can fix up at home. My friend, let's call him Bruce, felt such a call. His mission was to fix the professional malpractice of the Big Wheel people. His son needed one of those plastic three-wheel bikes, so Bruce scoured the toy stores but all he found were bikes with rotating wheels, no gadgets, lights, or sound.

So Bruce went to work. Long nights and endless requirements changes caused the usual challenges, and then there was the escalating budget. But after the holidays, Bruce reported a successful on-time Christmas delivery. A dream bike it was. This is what tricycles were meant to be. LCD readouts, turn signals (with multiple lights), speedometer, public address system, siren, and sound synthesizer – fortunately for the neighbors, lasers had yet to come down in price.

Bruce went on to produce another version of the bike, taking advantage of technology advances and of course, lessons learned. He offered his innovation to the manufacturer. They responded politely saying it was too complicated. Funny thing though, I came home recently and found a new tricycle in the garage. As I picked it up to move it, a stern woman's voice said, "This is mommy. Don't go near the street!" This was followed by the sound of a neighing horse. The tricycle was well into a cute little jingle about butterflies by the time I closed the door.

— Tony Henderson
Software Technology Support Center

MONTHLY COLUMNS:

ISSUE		COLUMN TITLE	AUTHOR
Issue 1: January	Publisher:	Best Training Includes Going Back to Basics	Lt. Col. G. A. Palmer
Back to Basics	BackTalk:	Pandora and the Magic Vase	R. Jensen
Issue 2: February	Publisher:	We've Come a Long Way From Machine Code to Current Programming Languages	H. B. Allgood
Programming Languages	BackTalk:	The First Book of EPP	G. Petersen
Issue 3: March	Publisher:	We Need the Right Tools for Quality Software	E. Starrett
The Case of Quality Software	BackTalk:	Did I Say "Koala Tea?"	D. A. Cook
Issue 4: April	Publisher:	Engineers at Their Best	T. L. Stauder
The People Variable	BackTalk:	A More Perfect Union	G. Petersen
Issue 5: May	Publisher:	Top 5 Contest Nominations Reveal Trends in COTS, E-Commerce, and Web Services	Lt. Col. G. A. Palmer
Strategies and Technologies	BackTalk:	Everybody Knows It's True	D. A. Cook
Issue 6: June	Publisher:	The Knowledge Flows Both Ways	H. B. Allgood
Commercial and Military Applications Meet	BackTalk:	Shock and Awe	G. Petersen
Issue 7: July	Publisher:	Top 5 Winners' Technologies Aim to Support the Warfighter: Several Used in Operation Iraqi Freedom	J. Jarzombek
Top 5	BackTalk:	Suggestion for "Bottom 5" Projects	D. A. Cook
Issue 8: August	Publisher:	Network-Centric Warfare Brings Increased Combat Power	Lt. Col. G. A. Palmer
Network-Centric Architecture	BackTalk:	Softbucks	G. Petersen
Issue 9: September	Publisher:	Managing Defects Together	T. L. Stauder
Defect Management	BackTalk:	Defect Mismanagement	D. A. Cook
Issue 10: October	Publisher:	Developers Meet a Variety of Complex Information and Data Sharing Needs	H. B. Allgood
Information Sharing	BackTalk:	CrossTalk Terminology Invitational	G. Petersen
Issue 11: November	Publisher:	Real-Time Software Development Requires Rigid Constraints	E. Starrett
Development of Real-Time Software	BackTalk:	Real Time - Military Style	D. Ludwig
Issue 12: December	Publisher:	Management Basics: A Necessary Foundation	T. L. Stauder
Management Basics	BackTalk:	Misbehaving Toys	T. Henderson

S T S C

Published by the
Software Technology
Support Center (STSC)

CrossTalk / MASE
6022 Fir Ave.
Bldg. 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737