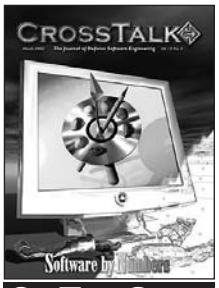
CROSSTALK

March 2002 The Journal of Defense Software Engineering Vol. 15 No. 3

Software by Numbers

Software by Numbers



On The Cover

Kent Bingham, Digital Illustration and Design, is a self-taught graphic artist/designer who freelances print and Web design projects.

Best Practices

Software Engineering Technology

Open Forum

Departments

CrossTalk

Sponsor	*Lt. Col. Glenn A. Palmer*
Publisher	*Tracy Stauder*
Associate Publisher	*Elizabeth Starrett*
Managing Editor	*Pamela Bowers*
Associate Editor	*Julie B. Jenkins*
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
CRSIP Online	www.crsip.hill.af.mil

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/TISE
7278 Fourth St.
Hill AFB, UT 84056-5205

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.

Getting the Numbers Out in the Open

When I was the lead of the Software Technology Support Center's measurement domain, numbers were always in high demand. I received numerous requests for examples of real numbers that other projects were collecting on their processes, quality, and return on investment. Unfortunately, most organizations consider these numbers to be extremely sensitive and are not willing to share them.

So, where do you look if you want to know how your organization is doing? First, you need to know just what it is you are trying to do; establish your organizational goals. Then ask the questions that will let you know whether or not you are meeting these goals. Finally, decide what information you need to answer these questions. One of the most important things with this approach is to start simple; five to 10 attributes (such as size, effort, schedule, cost, and quality) are a reasonable start. After you begin collecting this information, you can simply look to yourself to know how you are doing. Once you have some preliminary information, you can start to see how you are doing by comparing departments within the organization, by comparing the organization to itself via historical data, or by comparing your organization to the rest of industry. All of which brings us back to the coveted industry data.

The current lack of data is precisely why I am so excited about the articles featured in this month's theme section. Seldom does CROSSTALK receive articles that openly share such information. Donald J. Reifer's article, *Let the Numbers Do the Talking*, shares historical information from his collection of numerous projects. Reifer also includes some very good advice on how not to abuse the information.

In *How CMM Impacts Quality, Productivity, Rework, and the Bottom Line*, Michael Diaz and Jeff King share information that should prove useful for organizations trying to justify the cost of implementing a process improvement initiative. While actual numbers such as productivity were again considered sensitive, the actual process improvement numbers are very useful for providing justification for process improvement and potential return on investment. I am confident many organizations will benefit from this article.

Walt Lipke shares one way his organization uses and benefits from their numbers and passes this tip to CROSSTALK readers in *Statistical Process Control of Project Performance*.

As Bruce Allgood promised in February's "From the Publisher," our supporting articles begin with Suzanne Garcia's article, *Are You Prepared for CMMI?* This is an excellent article discussing information needed for successful technology transition. Next, Peter Amey shares information on an annotated subset of the Ada language developed in the United Kingdom that is showing success in his article *Correctness by Construction: Better Can also Be Cheaper*. Finally, Frank Richey shares ideas for helping the modeling and simulation community in *Modeling and Simulation CMMI: A Conceptual View*.

This month's articles show that measurements collected during software development can be a big help in predicting software development time, cost, and quality; can help justify funding for a new project; and can be used to monitor existing projects. Numbers are necessary for developing software.

When I received this month's theme articles, I was so excited to share their information that instead of trying to squeeze them into a planned issue as supporting articles, we created a new issue with this theme to highlight this information that was so often requested. I hope you find them useful for your planning and current practice comparisons.

Elizabeth Starrett
Associate Publisher

Let the Numbers Do the Talking

Donald J. Reifer
Reifer Consultants, Inc.

This article provides software cost and productivity benchmarks for 12 application domains that readers can use to determine how well their organization is doing relative to industry averages, and whether their software estimates are reasonable. In addition to addressing common questions raised relative to the benchmarks, this article summarizes the relative improvement that firms within the applicable industries are experiencing, which range from 8 percent to 12 percent annually.

For the past 30 years, I have been playing the numbers game. I have been developing benchmarks to confirm that the estimates I was developing were reasonable and achievable. To win the game, I have had to present the numbers in such a way that people I deal with would use them, not abuse them.

Everyone who works in the software business seems to be looking for numbers of one kind or another. I get at least one call a day asking me questions like, "What's the productivity that you've seen across the United States for software within the telecommunications domain?" or "What's the average cost/source line of code for a military system?"

My initial reaction is to try to avoid answering these questions. Why? Because I am afraid that whatever I say will be misquoted. Worse, I am afraid that the numbers I supply will be misused. When pressed for an answer, I express the number as a range. I qualify my answer by saying: " The average cost/line for a military system used for command and control varies between one and three hours per source line where an hour is expressed as directly chargeable labor, and a source line of code is defined using the Software Engineering Institute's counting framework as a logical line. Furthermore, the effort associated with this estimate is scoped to include requirements analysis, architectural design, development, and software integration and test tasks. It does not include system or beta testing, but does include support for requirements analysis."

"I have used the numbers to win the schedule and budget battles, to acquire investment dollars, ... and most importantly to win management's trust."

Sounds like a bunch of double-talk doesn't it? Well, it isn't. Nine out of 10 times, no matter what I say, the number is still misquoted or used out of context.

Needless to say, I am getting tired of being misquoted. In response, I have decided to write this article to put some of the more important numbers that I use in the public domain. Why? Well, for two reasons. First, I believe the community could use these numbers as benchmarks. They could serve as industry application domain norms against which organizations could compare their results to determine how well (or not so well) they are doing. Second, I want to get the community thinking about discussing and sharing "like" numbers. That is where I believe the real benefits lie.

Think about it. When push comes to shove, what really matters to management are the numbers. Again, let me give you an example. Suppose you are trying to get your bosses to invest in a new software-engineering environment. They will want justification. Typically, their decision whether or not to fund your proposal will revolve around whether money is available and the answers to the following questions: Will this investment save us money? If so, how much? What are the tax implications? Can we depreciate the equipment and software? If so, can we use either declining balance or straight-line depreciation schedules? What is the projected payback period and return on investment? Is this return higher than those who propose other alternative uses for the money?

Unfortunately, most of the engineers I have worked with during the years haven't the foggiest notion how to answer these questions. The net result is that their proposals are more often rejected than accepted because they fail to prepare a winning business case. However, the same engineers could improve their prospects of winning by using numbers to justify their proposals in terms of productivity improvement, cost reduction/avoidance, quality improvement, and/or time-to-market reduction strategies [1]. They could make the numbers sing to management.

Making Sense of the Data

My firm has been collecting cost, productivity, and quality data for more than two decades. These data are provided by organizations in exchange for benchmarks that they use for the following major purposes:

- Determine how the organization is doing relative to industry averages within an application domain (automation, command and control, telecommunications, etc.).
- Check the reasonableness of competitive bids.

Acronym List for Tables

IOC	Initial Operational Capability
IPT	Integrated Product Team
IRR	Internal Requirements Review
KSLOC	Thousands Source Lines of Code
LCA	Life Cycle Architecture (review)
LCO	Life Cycle Objectives (review)
MBASE	Model-Based Software Engineering
PDR	Preliminary Design Review
PRR	Product Readiness Review
SAR	Software Acceptance Review
SDR	System Design Review
SETD	Systems Engineering and Technical Direction
SLOC	Source Lines of Code
SM	Staff Month
SRR	Software Requirements Review
STR	Software Test Review
UTC	Unit Test Review

- Assess organizational shortfalls relative to the competition.

Occasionally, we publish snapshots of these databases [2]. When we do, we get lots of questions and feedback because interest is high, and the need for the numbers is great. To ensure the validity of the data submitted to us, we first screen and then normalize it using definitions that we have developed for that purpose. Once the numbers are entered into our databases, the data are checked for outliers and tested for homogeneity. Standard statistical regression techniques are then used to analyze the data and test it for sampling and other types of statistical errors.

The database has evolved to include data from approximately 1,500 projects. To maintain currency, none of the data retained in the database is more than 10 years old. We continuously refresh the database to purge older projects from it because they tend to bias the results of our analyses. We also remove outliers from the database because they tend to bias the results.

Over the years, my firm's databases have been the source of data for major software cost and productivity studies. For example, the U.S. Air Force relied heavily on the databases to develop a business case for its move to using Ada [3]. In addition, other organizations have used information from these databases to prepare knowledge bases and to calibrate their software cost models. Independent of the points Dr. Randall Jensen raised in his recent CROSSTALK article [4] on the topic of model calibration, we still believe and have the data to demonstrate that a calibrated cost model outperforms one that has not been calibrated. In addition, cost models seem much more accurate than activity-based costing done using Delphi approaches for large projects [5]. While critics of software cost models seem to abound [6], none of those throwing stones at the models have proposed alternative means that have higher prospects of accuracy.

First Look at Software Productivity

Software productivity refers to the ability of an organization to generate outputs (software systems, documentation, etc.) using the inputs or resources it has at its disposal (people, money, equipment, tools, etc.). Using this definition, an organization can increase its productivity by either focusing on the input or output part of the equation.

Application Domain	Number Projects	Size Range (KSLOC)	Avg. Productivity (SLOC/SM)	Range (SLOC/SM)	Example Applications
Automation	55	25 to 650	245	120 to 440	Factory automation
Command & Control	43	35 to 4,500	225	95 to 330	Command centers
Data Processing	36	20 to 780	330	165 to 500	Business systems
Environment/Tools	75	15 to 1,200	260	143 to 610	CASE tools, compilers, etc.
Military-Airborne	38	20 to 1,350	105	65 to 250	Embedded sensors
Military-Ground	52	25 to 2,125	195	80 to 300	Combat information center
Military-Missile	14	22 to 125	85	52 to 165	Guidance, navigation and control systems
Military-Spaceborne	18	15 to 465	90	45 to 175	Attitude control systems
Scientific	33	28 to 790	195	130 to 360	Seismic processing systems
Telecommunications	48	15 to 1,800	250	175 to 440	Digital switches and PABX
Trainers/Simulations	24	200 to 900	224	143 to 780	Virtual reality simulators
Web	64	10 to 270	275	190 to 975	Client/server sites
	500	10 to 4,500		45 to 975	

Table 1: *Software Productivity (SLOC/SM) by Application Domains*

Table 1 Notes

- The 500 projects taken from our database of more than 1,500 projects were completed within the last seven years by any of 38 organizations. (Each organizations' identity is anonymous due to the confidentiality of the data.)
- The scope of all projects starts from software requirements analysis and finishes with completion of software testing.
  - For military systems, the scope extends from software requirements review until handoff to the system test bench.
  - For Web systems, the scope extends from product conception to customer sell-off.
- This includes all directly chargeable engineering and management labor involved.
  - It includes programming, task management, and normal support personnel.
  - It does not include quality assurance, system or operational test, and beta test personnel.
- The average number of hours/staff month assumed was 152.
- SLOC is defined by Florac and Carleton [7] to be a logical source line of code using the conventions published by the Software Engineering Institute in 1993.
- Function point sizes were converted to SLOC using backfiring factors published by the International Function Point Users Group (IFPUG) in 2000, as available on their Web site.
- Different life-cycle models and methodologies are assumed. For example, Web projects typically followed a Rapid Application Development process and used lightweight methods, while military projects used more classical processes and methods.
- Different languages were used. For example, Web projects employed Java and Visual C while military projects used Ada and C/C++.

An input-based strategy would accentuate increasing workforce productivity via efficiencies gained by inserting better methods, tools, processes, facilities, and equipment and collaboration facilities. In contrast, an output-based strategy would place emphasis on reducing the amount of output required by using component technology, product lines, and architecture-centric reuse to eliminate a part of the work involved in developing the product.

Within many industries, productivity is commonly expressed as either source lines of code (SLOC)/staff month (SM) or function points (FP)/SM. Of course, the measures SLOC, FP, and SM must be carefully scoped and defined for these metrics to convey consistent meaning. In addition, there are many factors or cost drivers that cause each of these metrics to vary widely. These must be normalized when defining the terms. Because the Reifer Consultants, Inc. databases are primarily SLOC-based, we use this metric as the basis for our analysis. For those interested, we backfire the supplied FP data in our database using language conversion factors supplied by the International Function Point Users Group (IFPUG) to convert from FP to SLOC (e.g., one FP is expressed as so many lines of C++ or Java).

Table 1 summarizes the results of our analyses for 12 application domains for which we have collected data that we feel are of interest to the CROSSTALK community. The numbers in Table 1 were derived by taking a 500-project subset of our database and performing statistical analysis using various statistical tools. In addition, there are no foreign projects in the database to distort conclusions. Because our clients often challenge our numbers, we pay a great deal of attention to the tendencies and purity of our database. We cannot afford not to do this.

Application Domain	Ada83	Ada95	C/C++	3GL	Norm	Notes
Automation	*	*	30	45	30	Most implement ladder nets
Command & Control	70	*	50	100	75	
Data Processing	25	25	20	30	30	Most have moved to using Java and visual languages
Environment/Tools	25	*	25	30	25	
Military-Airborne	150	125	125	225	175	
Military-Ground	75	75	50	90	75	
Military-Missile	150	*	*	250	200	
Military-Spaceborne	150	*	150	200	175	
Scientific	75	*	65	85	75	
Telecommunications	50	35	40	80	55	Most use C/C++ and Unix
Trainers/Simulations	50	*	35	75	50	
Web	*	*	*	*	*	Most use Java and visual languages

* Not enough data

Table 2: *Software Cost ($/SLOC) by Language by Application Domain*

Table 2 Notes

- Dollars used to determine cost are assumed to be constant year 2000 dollars.
- The cost assumed per staff month of $12,000 assumes a labor mix and includes direct labor costs plus allowable overhead. This mix assumed that the average staff experience across the team in the domain was three years based on the average staff experience with the application and languages, methods, and tools employed to engineer it.
- Many languages were used in these applications; most projects used more than one language. For example, C/C++ was used heavily in the telecommunication domain, while Java was used extensively for Web applications.
- In addition, to keep costs down many organizations are trying to exploit commercial off-the-shelf packages and large legacy software systems written in languages like COBOL, FORTRAN, Jovial, and PL/1.

What About Software Cost Numbers?

While cost and productivity of software are related, they are separate considerations when dealing with numbers. To illustrate this point, I have seen several organizations increase their productivity and costs at the same time. In these cases, the organizations were very productive at generating software to the wrong requirements, or building and releasing products with lots of latent defects. That is why we focus attention on each set of numbers separately.

When analyzing our database, we find that software cost tends to be related to both the labor rates and language level (i.e., refers to the methods, tools, and language technology used by the project). To develop numbers of interest, we use $12,000 as the standard cost for a SM of effort exclusive of profit and general and administrative charges, as applicable. Table 2 shows the dollar cost per SLOC by application domains that we have developed as benchmarks.

How Is the Effort Distributed?

As most of us have learned, distribution of effort and schedule is a function of the life-cycle paradigm (i.e., a modeling method for the software process) selected for the project. In addition, as Fred Brooks [8] so nicely has explained, in many cases, effort and schedule cannot be interchanged. For the following three popular life-cycle paradigms, the allocations of effort and schedule are shown in the Tables 3, 4, and 5:

- Waterfall [9].
- Rational Unified Process (RUP) [10].
- Model-Based Software Engineering (MBASE) [11].

Formats in the tables vary because the numbers are based on slightly modified public references. In some cases, the allocations of effort (and schedule) are shown as ranges. In other cases, they are normalized so that they sum to 100 percent. In yet other cases, the sums are not normalized and therefore equal more than 100 percent.

Tables 3 and 5 clearly show the effort and duration to perform a particular activity relative to what is required for what they consider normal software development (e.g., excludes preparing software requirements and system test tasks). Table 4 reflects the effort and duration to perform tasks that are part of the RUP. Do not infer that MBASE takes 18 percent longer than RUP using these Tables. That is not the case because different life cycles embody different activities that make comparisons between them difficult and misleading. In any case, the results are what is important, not its format. If you are interested in more detailed comparisons between life cycles, see the appendices in [11], which provide the most complete coverage I have seen.

These allocation tables are very revealing. They tell us that software people do much more work than what is considered by most to be software development. This workload typically starts with analyzing the software requirements (i.e., these are typically developed by some other group like marketing or systems engineering) and ends with software integration and test. From Table 3, we can see that software people also take part in developing the requirements, which takes on average 7 percent additional effort and 16 percent to 24 percent more time, and they support system testing, which takes 12 percent more effort and 12.5 percent more time. This makes software cost estimates low by

Table 3: *Waterfall Paradigm Effort and Schedule Allocations*

Phase (end points)	Effort %	Duration %
Plans and Requirements (SDR to SRR)	7 (2 to 15)	16 to 24 (2 to 30)
Product Design (SRR to PDR)	17	24 to 28
Software Development (PDR to UTC)	52 to 64	40 to 56
Software Integration and Test (UTC to STR)	19 to 31	20 to 32
Transition (STR to SAR)	12 (0 to 30)	12.5 (0 to 20)
Total	107 to 131	116 to 155

Note: Percentage allocations for effort and duration are shown as ranges.

Table 4: *Rational Unified Process (RUP) Effort and Schedule Allocations*

Phase (end points)	Effort %	Duration %
Inception (IRR to LCO)	5	10
Elaboration (LCO to LCA)	20	30
Construction (LCA to IOC)	65	50
Transition (IOC to PRR)	10	10
Total	100	100

Note: Percentage allocations of effort and duration are normalized to 100 percent per the reference [10].

Table 5: *MBASE Paradigm Effort and Schedule Allocations*

Phase (end points)	Effort %	Duration %
Inception (IRR to LCO)	6 (2 to 15)	12.5 (2 to 30)
Elaboration (LCO to LCA)	24 (20 to 28)	37.5 (33 to 42)
Construction (LCA to IOC)	76 (72 to 80)	62.5 (58 to 67)
Transition (IOC to PRR)	12 (0 to 20)	12.5 (0 to 20)
Total	118	125

Note: Percentage allocations of effort and duration are shown as ranges and are not normalized per the reference [11].

at least 19 percent and schedule estimates short by about 28.5 percent. It is no wonder that software people feel shorted when the budgets are allocated.

What About the Other Support Costs?

Let us look at the typical costs software organizations spend supporting other engineering organizations. For example, they might participate as members of some integrated product team tasked with developing requirements for the project. Because this effort takes time and effort, funds to perform the work involved need to be estimated, budgeted, and controlled.

As another example, software people will be called upon to assist during systems test and evaluation. They might be called upon to either conduct software beta testing or fix software, hardware, or systems problems during bench, system, and/or operational testing. That is where software shines. It is used to make the system work.

What is important is that these activities consume effort and take talent away from the mainline software development tasks. That means all of these activities need to be thoroughly planned, estimated, budgeted, scheduled, staffed, directed, controlled, and managed by the software manager as the project unfolds.

There is a lot of controversy over how much effort is needed to perform different types of support tasks. Based on experience [12], Table 6 shows examples of how much effort is expended in providing needed support expressed as an average and a range.

Because Independent Verification and Validation and System Engineering Technical Direction contractors increase the development contractor's workload in military contracts, they require additional effort to support such relationships. This is especially true when military organizations use a federally funded research and development contractor to perform system integration and task-direction type tasks.

Support Cost Category	Effort (% of software development costs)	Notes
Requirements synthesis and IPT participation	10 (6 to 18)	Participation in systems definition and specification activities.
Systems integration and test	30 (0 to 100)	Supporting problem resolution activities as the system is integrated and tested.
Repackaging documentation per customer requirements	10 (0 to 20)	Repackaging documentation to meet some customer preference instead of that dictated by the approved organizational process.
Formal configuration management (CM)	5 (4 to 6)	Support to system level CM activities (version control & support for the Software Change Control Board are already included in the software estimate).
Independent software quality assurance (SQA)	5 (4 to 6)	SQA when done by a separate organization.
Independent Verification & Validation or SETD support contractor	6 (4 to 10)	Development organization support to an independent contractor hired to perform technical oversight and provide direction.
Total	66 (18 to 160)	

Table 6: *Typical Software Support Costs*

Table 6 Notes

- Percentage allocations of effort are shown as both an average and a range in parentheses.
- Version control of software deliverables is included in our other numbers as is normal quality assurance activities. The additional effort illustrated by Table 6 refers to formal activities like project-level change-control boards that organizations must support.
- If there are subcontracts involving software, additional effort must be added to Table 6 to provide for software support in this area.
- If large amounts of commercial off-the-shelf software will be used, additional effort must be added to Table 6 to support the evaluation, glue code development, integration, and other life-cycle tasks that the software organization will have to support.
- Percentage expressed uses software development costs as its basis. For example, the chart states that the cost on average quoted for a software job should be 166 percent of the base (the base plus 66 percent) to cover the additional costs associated with the support identified in Table 6 when all the categories listed are applicable.
- These support costs can vary greatly based on the manner in which the firm is organized and the work allocated. For example, some firms combine their CM and SQA support in a single Product Assurance organization. In such cases, the average cost for both is 7 percent (6 percent to 8 percent) because they take advantage of economies of scale.
- System integration and test support do not include operational test and evaluation for military projects and beta testing for commercial projects. These can require even more support be applied than that identified. For example, we have seen aircraft projects that have gone through extensive flight testing burn thousands of hours of software support during two- and three-year time periods. As another example, we have seen commercial projects also burn thousands of hours supporting beta testing at remote user sites.
- Many firms have adopted defined processes at the organizational level and are rated as a Level 3 using the Software Engineering Institute's Software Capability Maturity Model® (SW-CMM®) [13]. When this is true, these firms generate documentation as a normal part of their engineering processes. If the customer wishes to reformat this documentation, there will be a repackaging cost because this is not the normal way these firms conduct their business.

Have We Made Progress?

It is also interesting to look at the trends associated with productivity and cost. Based on the data we analyzed, the nominal improvement firms experience across industries is from 8 percent to 12 percent a year. Those who invest more, typically gain more. For example, jumping a single CMM level can reduce software development costs from 4 percent to 11 percent based on a recent Ph.D. dissertation done by Brad Clark at the University of California [14].

Because I plan to write a separate article on this topic in the near future, I will not clutter this article with additional trend data. However, it is important. For example, a 10 percent improvement in productivity can be used to justify a multi-million dollar software initiative for firms with 500 or more software engineers.

Call to Action

I encourage those of you with solid numbers to throw stones. One of my goals in writing this paper is getting the community to think about, discuss, and share like numbers. If you have more concrete numbers, I encourage you to put them into the public domain. Challenge my assumptions and summarize your experience so the community can use it. However, if you do, please tell the community how you derived the numbers, what their source is, and how you normalized them. Do not just throw

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

stones at my results. That would be counter productive. Share your experience with others and help the community to develop realistic benchmarks that they can use for comparison purposes.

I really would encourage those who do not have numbers to develop them. Throughout my entire career, I have used the numbers to win the schedule and budget battles, to acquire investment dollars, to improve my decision-making abilities, and most importantly to win management's trust. I gained credibility with management at all levels of the enterprise by discussing both the technical and business issues associated with my proposals. I was successful in getting approvals because I emphasized business goals and showed management that what I was proposing made good business and technical sense. It is not surprising that I am a strong advocate of managing by the numbers. Try it, and I think you will like it.◆

References

1. Reifer, D. J. Making the Software Business Case: Improvement by the Numbers. Addison-Wesley, 2001.
2. Reifer, D. J., J. Craver, M. Ellis, and D. Strickland, eds. "Is Ada Dead or Alive Within the Weapons System World?" CROSSTALK Dec. 2000: 22-24.
3. Ada and C++: A Business Case Analysis. U. S. Air Force, 1991.
4. Jensen, R. "Software Estimating Model Calibration." CROSSTALK July 2001: 13-18.
5. Reifer, D. J. "Comparative Accuracy Analysis of Cost Models to Activity-Based Costing for Large Scale Software Projects." Reifer Consultants, Inc., 1996.
6. Ferens, E., and D. Christensen, eds. Calibrating Software Cost Models to Department of Defense Databases – A Review of Ten Studies. Air Force Research Laboratories, Feb. 1998.
7. Florac, W. A., and A. D. Carleton, eds. Measuring the Software Process. Addison-Wesley, 1999.
8. Brooks, F. The Mythical Man-Month, Anniversary Edition. Addison Wesley, 1995.
9. Boehm, B. W., C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, eds. Software Cost Estimation with COCOMO II. Prentice-Hall, 2000.
10. Kruchten, P. The Rational Unified Process. Addison-Wesley, 1998.
11. Royce, W. Software Project Management: A Unified Framework. Addison-Wesley, 1998.
12. Reifer, D. J. A Poor Man's Guide to Estimating Software Costs. 8th ed., Reifer Consultants, Inc., 2000.
13. Paulk, M. C., C. V. Weber, B. Curtis, and M. B. Chrissis, eds. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, 1995.
14. Clark, B. "Quantifying the Effects on Effort of Process Improvement." IEEE Software Nov./Dec. 2000: 65-70.

About the Author

Donald J. Reifer is one of the leading figures in the fields of software engineering and management, with more than 30 years of progressive experience in government and industry. In that time, he has served as chief of the Ada Joint Program Office and the director of the Department of Defense Software Reuse Initiative. He is currently the president of Reifer Consultants, Inc., which specializes in helping clients improve the way they do business. Reifer's many honors include the American Institute of Aeronautics and Astronautics Software Engineering Award, the Secretary of Defense's Medal for Outstanding Public Service, the NASA Distinguished Service Medal, the Frieman Award, and the Hughes Aircraft Fellowship. Reifer has more than 100 publications, including *Software Management Tutorial* (6th edition) and *Making the Software Business Case: Improvement by the Numbers.*

P.O. Box 4046
Torrance, CA 90505
Phone: (310) 530-4493
Fax: (310) 530-4297
E-mail: d.reifer@ieee.org

WEB SITES

Earned Value Management

www.acq.osd.mil/pm

Sponsored by the Office of the Under Secretary of Defense (Acquisition, Technology, and Logistics) Acquisition Resources and Analysis/Acquisition Management, the Earned Value Management Web site provides information on earned value project management (EVM) for government, industry, and academic users. Find current editions of policy documents, information on government EVM contacts and International Performance Management Council members, speeches, training material, EVM software, supplier links, frequently asked questions, and more.

Project Management Institute

www.pmi.org

The Project Management Institute (PMI) claims to be the world's leading not-for-profit project management professional association, with more than 86,000 members worldwide. PMI establishes project management standards, provides seminars, educational programs, and professional certification for project leaders.

Defense Contract Management Agency

www.dcma.mil

The Defense Contract Management Agency is the Department of Defense contract manager, responsible for ensuring federal acquisition programs, supplies, and services are delivered on time, within cost, and meet performance requirements. The agency is currently conducting in-plant observations using the Capability Maturity Model® to deploy a standard methodology via continuous process evaluations of contractors. Details concerning the process, responsibilities, and outcomes are captured in the "Method Description Document" available on the Web site.

Software Technology Support Center

www.stsc.hill.af.mil

The Software Technology Support Center (STSC) is an Air Force organization established to help other U.S. government 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.

How CMM Impacts Quality, Productivity, Rework, and the Bottom Line

Michael Diaz and Jeff King
General Dynamics Decision Systems

The Software Engineering Institute's Capability Maturity Model® (CMM®) plays a major role in defining software process improvement (SPI) in many companies. The question of cost/benefit has come up frequently in organizations contemplating SPI activities. This article will explore various cost/benefit issues and examine performance results of various General Dynamics Decision Systems' projects with relation to its software process maturity. The quantitative data presented indicates CMM-based improvement yields dividends in terms of higher productivity and software quality. Each level of improvement significantly cuts defect density, improves productivity, and reduces rework.

General Dynamics Decision Systems supplies communications and information technology for military and government customers and employs approximately 1,500 engineers that design and build a wide variety of government electronic systems. Approximately 360 engineers are directly involved in software development.

The question of cost/benefit has come up frequently in organizations contemplating software process improvement (SPI) activities. This article will explore various cost/benefit issues and examine performance results of various General Dynamics Decision Systems' projects with relation to its software process maturity. It also discusses the implementation strategies put in place to achieve process improvement and other organizational goals; some "lessons learned" about process improvement are also presented.

CMM Overview

The Software Engineering Institute's (SEI) Capability Maturity Model® (CMM®) [1] plays a major role in defining SPI in many companies. The CMM consists of five levels of process maturity where each level has an associated set of key process areas (KPAs). At the initial Level 1 maturity, software projects rely on the skills and heroic efforts of individual engineers. There are no KPAs associated with Level 1. Firefighting is prevalent and projects tend to leap from one emergency to the next.

CMM Level 2, the Repeatable maturity level, has six KPAs associated with it. These KPAs relate to requirements management, project planning, project tracking and oversight, subcontract management, quality assurance, and configuration management. Projects under a Level 2 organization are repeatable and under basic management control.

At the Defined Level 3 maturity level, the software development organization now defines common processes, develops training programs, focuses on intergroup coordination, and performs peer reviews. The result is the development of tailorable software processes and other organizational assets so that there is a certain level of consistency across projects.

"Peer reviews have been widely recognized in the industry for being the single most important factor in detecting and preventing defects in software products."

At the Managed Level 4 maturity level, the software development organization implements a quality and metric management program and monitors both project and organizational performance (i.e., establishes software process capability and variance measures using statistical process control charts).

At the Optimizing Level 5 maturity level, quantitative data are used for process improvement and defect prevention. In addition, technology changes are introduced and evaluated in an organized, systematic process.

Summary Results

General Dynamics Decision Systems has three software-engineering organizations: Integrated Systems, Information Security Systems, and Communication Systems. As of Nov. 16, 2001, all three had been externally assessed at CMM Level 5 using the CMM-Based Appraisal for Internal Process Improvement (CBA IPI). Our metrics and historical repository contain data from past Level 2, 3 and 4 programs within the Information Security Systems engineering organization, as well as our current Level 4 and 5 programs.

Rework, phase containment, quality, and productivity metrics are based upon history as well as the current measures of approximately 20 programs, each at various stages of the software life cycle.

At General Dynamics Decision Systems, every project performs a quarterly SEI self-assessment. The project evaluates each KPA activity with a score between one and 10, which is rolled up into an average score for each KPA. Any KPA average score falling below seven is determined to be a weakness. The SEI level for the project is defined as the level in which all associated KPAs are considered strengths, i.e., all KPA average scores are seven or above.

Table 1 summarizes the General Dynamics Decision Systems' improvement trends for rework, phase containment, predicted quality, and productivity by CMM level. Percent rework is a measure of the percentage of the project development time that was expended due to rework. Phase containment

Table 1: *General Dynamics Decision Systems Project Performance Versus CMM Level*

CMM Level	Percent Rework	Phase Containment Effectiveness	CRUD Density per KSLOC	Productivity (X Factor Relative)
2	23.2%	25.5%	3.20	1 x
3	14.3%	41.5%	0.90	2 x
4	9.5%	62.3%	0.22	1.9 x
5	6.8%	87.3%	0.19	2.9 x

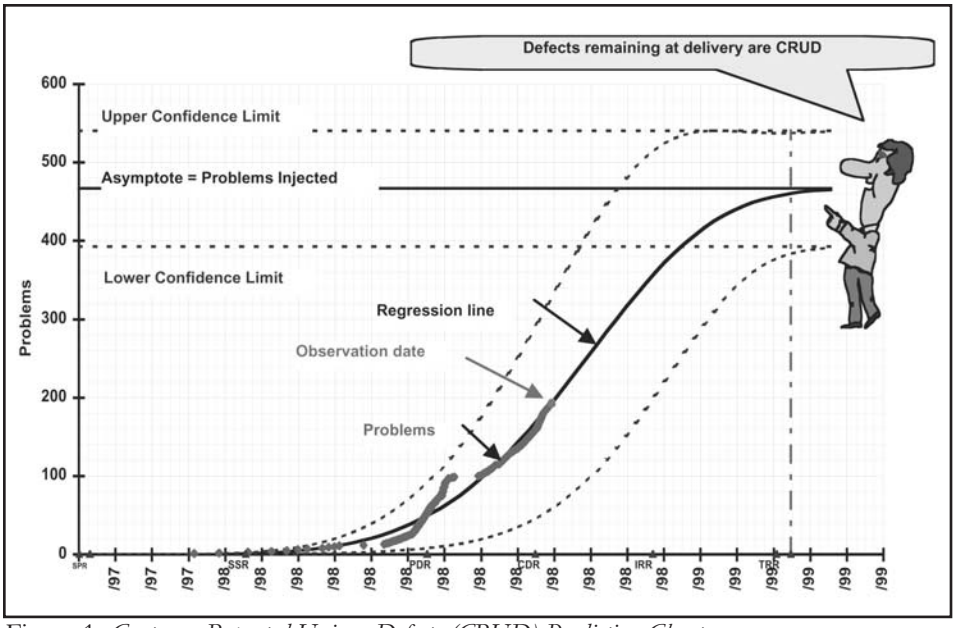


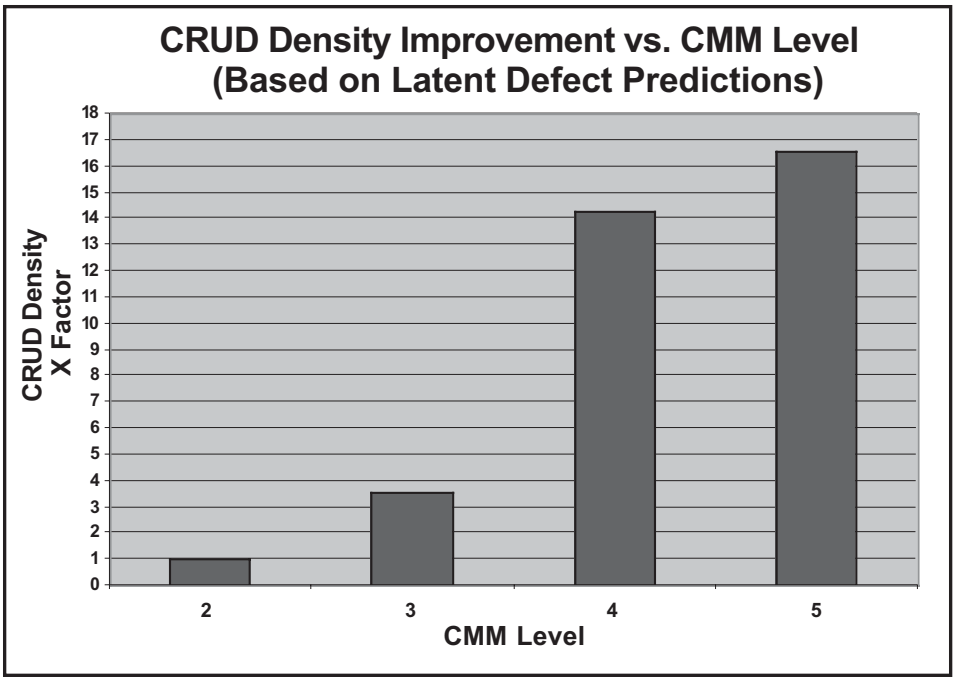
Figure 1: *Customer Reported Unique Defects (CRUD) Prediction Chart*

effectiveness is a measure of defect containment within the phase in which it was created. Higher phase containment is equivalent to early detection of defects within the same phase in which it was created. Predicted quality is defined as the number of latent defects or Customer Reported Unique Defects (CRUD) per thousand source lines of code (KSLOC). Productivity is displayed in X factor terms that are defined as the productivity average of all programs within a certain CMM level divided by the productivity average of all Level 2 programs. The quality, rework, and productivity performance for each program is obtained from General Dynamics Decision Systems' internal metrics and categorized by CMM level as determined by each project's internal self-assessment.

Detailed Metric/Results Analysis

This section will discuss each General Dynamics Decision Systems' metric collected (i.e., percent rework, phase containment, quality, and productivity) and discuss the improvement results with relation to CMM maturity level. As will be discussed later, specific improvement results are not entirely attributable to increasing CMM maturity levels since the organization has put into place initiatives in cycle time and quality improvement above and beyond the SEI CMM.

Figure 2: *Quality Versus CMM Level*



1. Quality Metric

At General Dynamics Decision Systems, post release software quality is defined as the number of predicted latent defects per thousands of delivered source instructions. Latent defects are predicted based upon the rate of new problems discovered during development. The total number of problems introduced in a software product is the sum of problems detected during development and the latent defects remaining at product release.

The total number of software problems introduced in a software product can be estimated by using historical defect density data from similar projects and tracking problems found early in the development cycle. A method to predict problems throughout the development cycle is given by Arkell and Sligo in "Software Problem Prediction and Quality Management [2]." Latent defects or CRUD are defects in the delivered product. CRUD is not experienced during software product development but it can be estimated.

Future problems can be predicted from the pattern of problems already detected by examination (peer review or inspection), by testing, and by using work products already examined or tested. The cumulative number of problems detected over time tends to follow an S-shaped curve.

The number of problems remaining to be discovered (CRUD) is depicted in Figure 1 as the difference between the asymptote (total problems injected) and the regression line at the point of delivery (the dashed vertical line after TRR).

CRUD is an indicator of software quality. Since it is reasonable for small products to have less CRUD than large products, General Dynamics Decision Systems uses CRUD density (CRUD per KSLOC) as one of the software quality indicators.

Quality Results

Figure 2 examines the predicted quality improvement as projects progress up the various SEI CMM levels. This chart shows that predicted quality improves (which is synonymous with decreasing latent defect [CRUD] predictions) with increasing SEI CMM levels.

Quality Analysis

The metric data show that compared with an average Level 2 program, Level 3 programs have 3.6 times fewer latent defects, Level 4 programs have 14.5 times fewer latent defects, and Level 5 programs have 16.8 times fewer latent defects.

The improvement in quality is expected for projects that transition from Level 2 to 3 due to the Peer Review KPA found in Level 3. Peer reviews are widely recognized in the industry for being the single most important factor in detecting and preventing defects in software products. Quality is also expected to improve for projects transitioning from Level 3 to 4 due to the Quantitative Process Management and Software Quality Management KPAs. Using quality metric data such as peer review effectiveness and phase containment effectiveness will allow the project to modify its processes when the observed metric falls below the organizational and project control limits.

The improvement from Level 4 to Level 5 is attributable to the Defect Prevention and Process Change Management KPAs. Projects operating at this level perform Pareto analysis on the root cause of their problems and perform causal analysis to determine the process changes needed to prevent similar problems from occurring in the future.

It should be noted that large improvements in defect density are more readily obtained when the number of defects is large, as would be expected in lower maturity level projects. At higher maturity levels, it becomes more and more difficult to dramatically reduce the defect density.

2. Phase Containment Metric

Phase containment is defined as the ratio of problems detected divided by the number of problems inserted within a phase. For example, if 100 problems were introduced in detailed design but only 75 problems were detected by the peer-review process, then the phase containment effectiveness would be 75 percent. The General Dynamics Decision Systems' goal is at 85 percent phase containment effectiveness. Projects below this threshold perform causal analysis to improve their peer review and testing processes. The focus in improving phase containment is to catch problems as early as possible. The cost of fixing problems escalates dramatically the longer the problem remains undetected in the software life cycle.

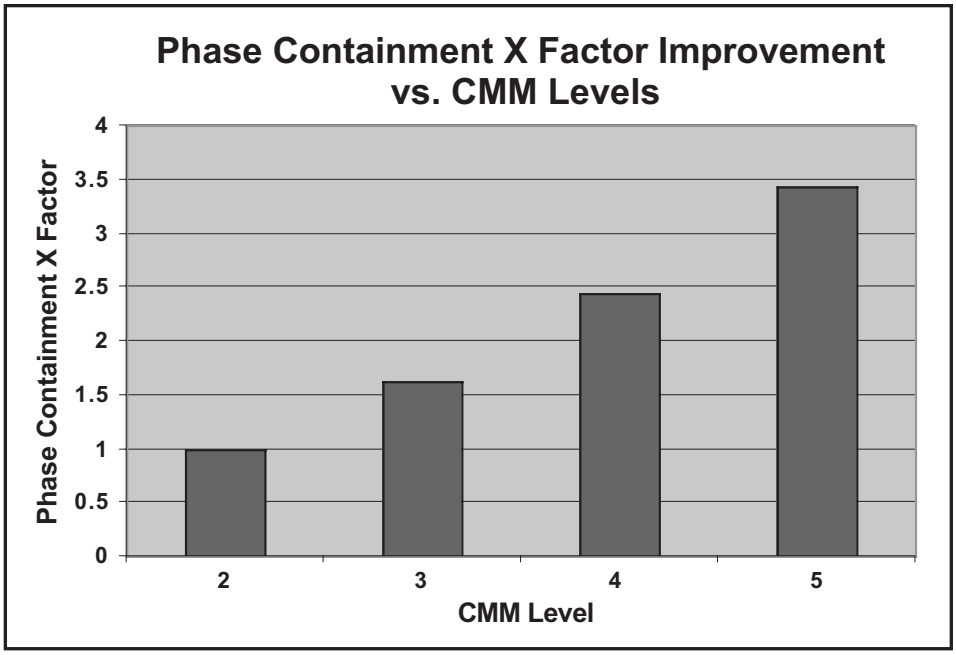


Figure 3: *Phase Containment Effectiveness Versus CMM Level*

Phase Containment Results

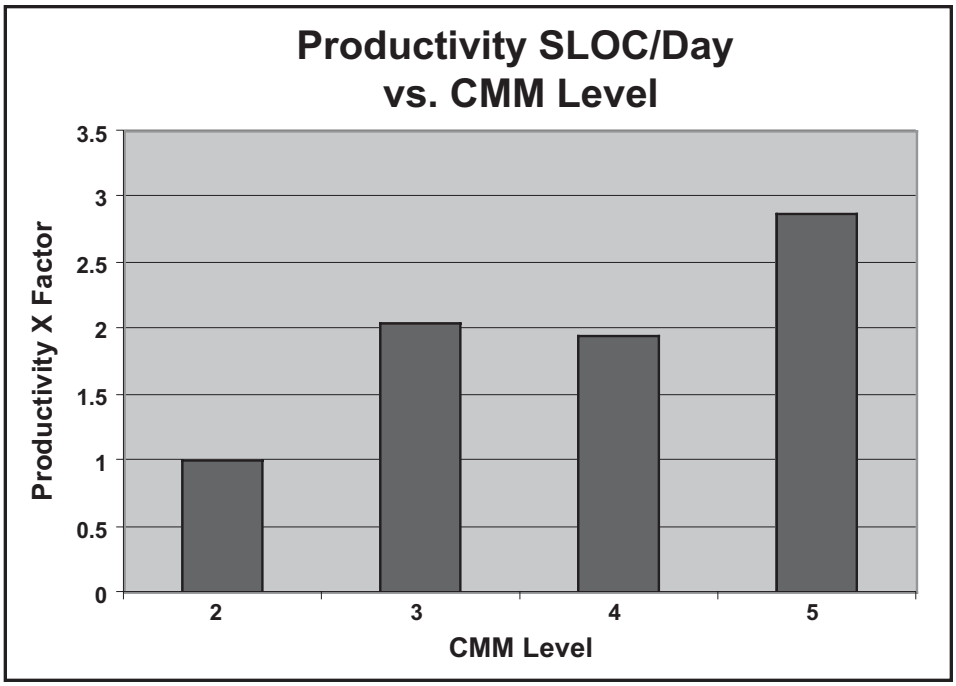
Figure 3 illustrates the Phase Containment Effectiveness improvements with respect to CMM level.

Phase Containment Analysis

Analysis of the data shows that compared with an average Level 2 program, Level 3 programs have 1.6 times better phase containment effectiveness, Level 4 programs have 2.4 times better phase containment effectiveness, and Level 5 programs have 3.4 times better phase containment effectiveness.

The improvement in phase containment effectiveness from Level 2 to 3 is primarily due to the Peer-Review KPA. Improvements from Level 3 to 4 are due to increased attention on peer review effectiveness using statistical process control charts and monitoring and removing assignable causes of variation, e.g., variation not inherent in the peer-review process. Improvements from Level 4 to Level 5 are attributable to the increased focus on the Defect

Figure 4: *Productivity Versus CMM Level*



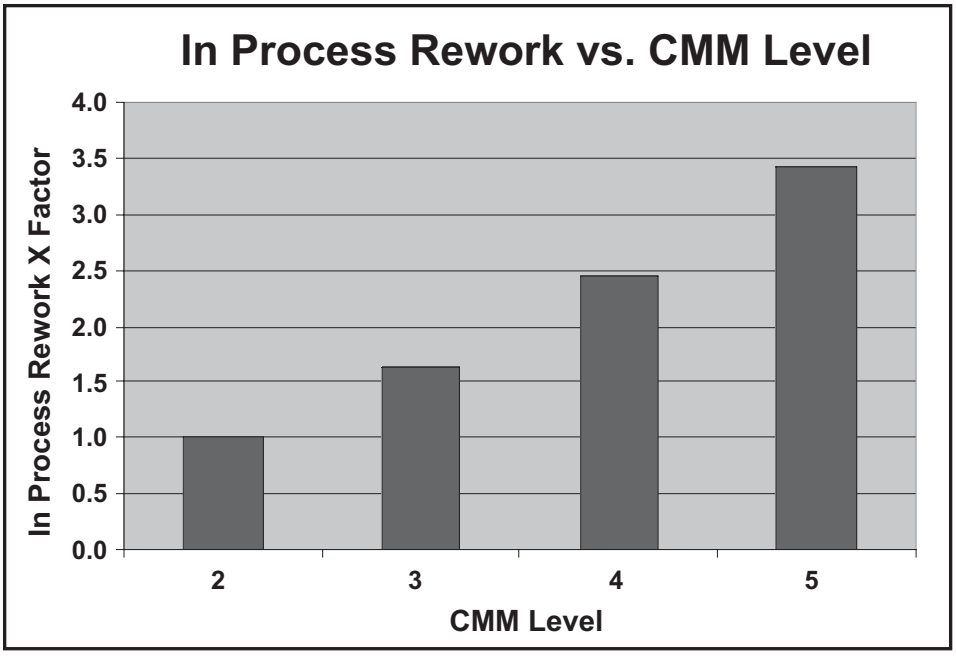


Figure 5: *In Process Rework Versus CMM Level*

Prevention and Process Change Management KPAs.

3. Productivity Metric

Productivity is defined as the amount of work produced divided by the time to produce that work. This may be measured in SLOC per hour, or some similar measure. Each project at General Dynamics Decision Systems tracks its productivity by measuring SLOC produced and the number of hours to produce that code.

Productivity Results

For proprietary reasons, the actual number of lines of code per hour is not shown; however, the relative productivity between projects at different levels of maturity can be seen in Figure 4 (see page 11). The data are normalized to the productivity of an average Level 2 project. The X factor is defined as the relative improvement as compared with a Level 2 program. For example, if a Level 2 program has an average productivity of eight SLOC per day and a Level 3 program has an average productivity of 16 SLOC per day, one could say that the Level 3 programs have a 2 X factor as compared with Level 2 programs.

Productivity Analysis

Project data show that compared with an average Level 2 program, Level 3 programs show a 2 X factor improvement in productivity, Level 4 programs show a 1.9 X factor improvement in productivity and Level 5 programs show a 2.9 X factor improvement in productivity.

Productivity is affected by factors other than process maturity, most importantly technology changes. For example, the data shown include projects that may have started before some form of automated code generation became available. In addition, the amount of code reuse on a project can greatly affect the productivity of that project. As projects increase their level of maturity, the ability to effectively reuse software source code is enhanced. Likewise, software code that is reused from a high maturity level project requires less rework and is more easily understood. These factors act as multipliers in the productivity of high maturity level projects.

It is interesting to note that in the transition from Level 3 to Level 4, projects do not experience a statistically significant change in productivity. This appears to be a side effect of Level 4 being a transitionary state in which projects quickly progress to Level 5 practices once making Level 4. Level 4 programs can monitor their critical processes using statistical process control techniques. However, the skills needed to perform causal analysis and process change are obtained at Level 5.

It is also possible that the effective utilization of statistical process control and software quality management introduced at Level 4 becomes more effective over time and that benefits are not realized in the short term in this transitionary state. As with any new technology, it is expected that a cycle of absorption is needed before the full benefits can be observed.

Table 2: *Model of Rework Costs per CMM Level for 100 KSLOC Program*

CMM Level	CRUD for 100 KSLOC	Post Release Rework (hrs)	Pre Release Rework (hrs)
Level 5	19.11567615	306	2,397
Level 4	22.0952381	354	3,358
Level 3	88.25757576	1,412	5,043
Level 2	315.8653846	5,054	8,208

4. Rework Metric

At General Dynamics Decision Systems, each software engineer enters his or her time card on a daily basis to include hours expended, charge number, and burden code. Burden codes measure major process activities and are subdivided into the following categories:

G - Generate
V - eValuate
K - reworK
S - Support

We measured the percentage of rework on a project as defined by "total hours for rework divided by total hours on the project." We then evaluated project results based upon CMM levels. Note that post-release rework due to maintenance is not included in this analysis.

Rework Results

The amount of rework is normalized to a Level 2 program where rework reduction is shown as an improvement in Figure 5. The X factor rework reduction is calculated by taking the percentage of rework for Level 2 programs and dividing the percentage of rework for all projects with a given CMM level.

As compared with an average Level 2 program, Level 3 programs show a 1.6 X factor reduction in rework, Level 4 programs show a 2.4 X factor reduction in rework, and Level 5 programs show a 3.4 X factor reduction in rework. It is interesting to note that the rework X factor improvements match the phase containment X factor improvements. This is not surprising due to the correlation of early in-process fault detection

to the amount of reduced rework, i.e., better phase containment directly relates to lower rework.

Process Improvement Implementation Strategies

The following strategies are a result of several lessons learned from the software process improvements made at General Dynamics Decision Systems:

- Plan for organizational software process focus and definition impacts during reorganization planning.
- Statistical process control training and training on assignable causes of variation.
- Focus on new projects. It is extremely difficult to change projects, especially at a low maturity level, once they have started.
- A top down focus is essential before getting buried in the details of the CMM; start with the intent of each KPA and determine how it fits into your environment.
- Emphasize productivity, quality, and cycle time. Avoid process for the sake of process.
- Management commitment is needed from all levels; commitment from upper management won't be enough unless individual project leaders/managers are also determined to succeed.
- Practitioners and task leaders, not outside process experts, should be used to define processes.
- Managers need to be convinced of the value of process improvement; it's not free, but in the long run it certainly pays for itself.
- Copying process documents from other organizations usually does not work well; the process must match your organization.
- Overcoming resistance to change is probably the most difficult hurdle when climbing the CMM ladder.
- There are no silver bullets! Process change takes time, talent, and a commitment with which many organizations are uncomfortable. If it was easy, everyone would have already done it.

Return on Investment

The process improvement efforts to support 360 software engineers include the following:

- Full-time chief software engineer and metrics champion.
- Weekly software improvement meetings by software task leaders.
- Project kickoffs, phase-end reviews, and post mortems.
- Focused process improvement working groups by project personnel.

The above process improvement efforts were approximately 2.5 percent of the base staffing of 360 software engineers. Given the following assumptions for a single project:

- 100 KSLOC size project for two years.
- Sixteen hours to fix a defect found after software release.

The post and pre-release rework calculated from our return-on-investment (ROI) model is shown in Table 2.

Assuming 2.5 percent investment for process improvement on a project to progress one level of CMM maturity within one year, the ROI per CMM level is depicted in Table 3.

The ROI calculations do not take into account the added benefit of being able to apply existing resources for pursuit/execution of new business opportunities due to improved cycle times and earlier completion dates.

CMM Level Transition	Cost for SPI in hrs (2.5% of Base)	Cost Savings on Rework (hrs)	Return on Investment
Level 4 to 5	884	1,009	14%
Level 3 to 4	1,310	2,744	109%
Level 2 to 3	2,544	6,806	167%

Table 3: *Return on Investment (ROI) by Level Transitions*

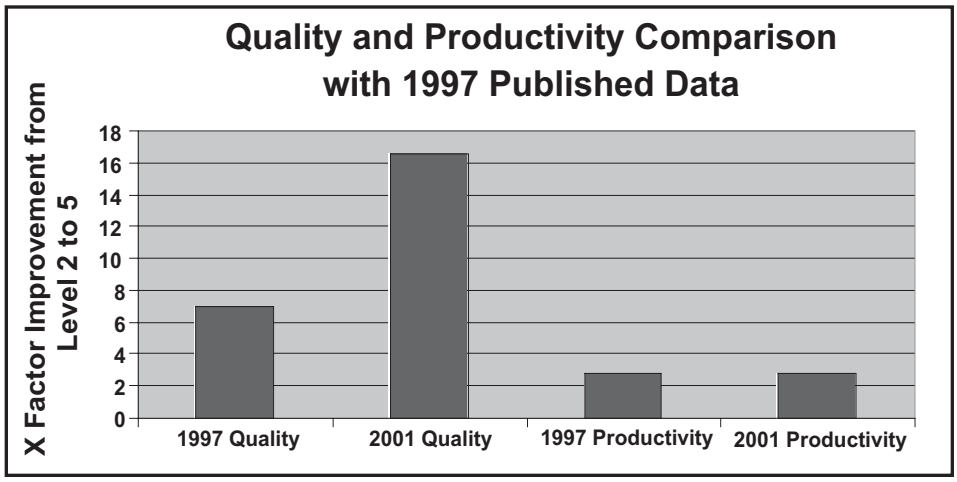
Conclusion

Similar to the earlier 1997 results published regarding General Dynamics Decision Systems' Software Process Improvement [3], each level of software maturity results in improved quality and productivity. Each level of CMM maturity reduces defect density by a factor of almost four on the average until Level 4 is reached, where a 16 percent improvement is seen from Level 4 to 5. Phase containment effectiveness and rework improve on the average by 50 percent with each maturity level. Productivity improves 100 percent for Level 2 to 3 transitions and by 50 percent for Level 4 to 5 transitions.

Comparing the General Dynamics Decision Systems' 2001 study with the published 1997 results shows some similarities and some differences as shown in Figure 6. The productivity improvement from Level 2 to 5 is about the same for both studies, around a 2.8 X factor improvement. The quality improvements, however, are more pronounced, 16.5 X factor improvement between Levels 2 and 5, than the General Dynamics Decision Systems' 1997 study that documented a 7 X factor improvement between Levels 2 and 5 [3]. This suggests that the quality benefits increase the longer an organization is able to maintain a Level 5 maturity capability.

The ROI analysis shows the largest benefit is going from Level 2 to 3 with 167 percent ROI. Level 3 to 4 advancement also shows a significant 109 percent ROI. Although the Level 4 to 5 ROI of 14 percent is not as significant as the other level transitions, subjective experience from these authors indicates that Level 4 projects are transitionary and short lived, quickly obtaining Level 5 much earlier than the one year per level

Figure 6: *Quality and Productivity Comparison with 1997 Published Data*



Coming Events

March 25-28
Software Test Automation Conference
San Jose, CA
www.sqe.com/testautomation

April 8-10
Secure E-Business Executive Summit
Arlington, VA
www.secure-biz.net

April 9-10
Southeastern Software Engineering Conference
Huntsville, AL
www.ndia-tvc.org/SESEC2002/

April 29–May 2
Software Technology Conference 2002 "Forging the Future of Defense Through Technology"
Salt Lake City, UT
www.stc-online.org

May 13-17
Software Testing Analysis and Review (STAREAST 2002)
Orlando, FL
www.sqe.com/stareast

June 3-6
Combat Identification Systems Conference
Colorado Springs, CO
www.usasymposium.com

July 22-25
Joint Advanced Weapons Systems Sensors, Simulation, and Support Symposium (JAWS S3)
Colorado Springs, CO
www.jawswg.hill.af.mil

transition assumption used by the analysis model.

These data suggest that Level 5 is the most desirable state an organization would strive for in order to maximize the quality and productivity performance of a project.

Process improvement takes time to institutionalize and requires a commitment from management in order to succeed. Achieving higher levels of process maturity requires an investment of time and money in process improvements, including tool integration to aid in the collection and interpretation of quantitative data.

In conclusion, process improvement activities must be undertaken with a look at return on investment. Higher maturity organizations take this into account when initiating SPI activities. The CMM by itself does not assure improved performance results. Performance improvement must be specifically identified as the goal for SPI to avoid process for process sake. Tailoring of processes and a focus on cycle time are needed in addition to the traditional CMM emphasis.◆

References

1. Paulk, Mark, et al. "Capability Maturity Model Version 1.1." IEEE Software July 1993: 18-27.
2. Arkell, Frank, and Joseph Sligo, eds. "Software Problem Prediction and Quality Management." Conference Proceedings of the Seventh International Conference on Applications of Software Measure-ment, Oct. 1996.
3. Diaz, Michael, and Joseph Sligo, eds. "How Software Process Improve-ment Helped Motorola." IEEE Software Sept. 1997: 75-81.

About the Authors

Mike Diaz is a chief software engineer for the General Dynamics Decision Systems and is responsible for all aspects of software development in an organization of 360 software engineers. Diaz was a key contributor leading to General Dynamics Decision Systems' second Capability Maturity Model® Level 5 rating. Diaz's experience includes 19 years of software technical leadership in requirements management, systems engineering, security architectures, and secure key management systems while at General Dynamics Decision Systems. Diaz has been awarded membership in General Dynamics Decision Systems' Scientific Advisory Board Association, the highest technical association within General Dynamics Decision Systems. Diaz received a bachelor's of science degree in electrical engineering and a master's degree in computer engineering from Boston University.

General Dynamics Decision Systems
8220 E. Roosevelt St.
Scottsdale, AZ 85252
Fax: (480) 675-2398
E-mail: michaeldiaz@gd-decision systems.com

Jeff King is a senior recruiter for General Dynamics Decision Systems in Scottsdale, Ariz. He has worked for General Dynamics Decision Systems for six years and has 15 years total experience in technical recruiting. During his career, King has supported information technology, telecommunications, aerospace, and defense industries. He received a bachelor's degree in economics from Northern Arizona University in Flagstaff, Ariz. in 1979 and earned 60 hours post graduate at Arizona State University.

General Dynamics Decision Systems
8220 E. Roosevelt St.
Scottsdale, AZ 85252
Phone: (480) 675-2995
E-mail: jeff.king@gd-decision systems.com

Statistical Process Control of Project Performance

Walt Lipke
Oklahoma City Air Logistics Center

With today's increased emphasis on statistical process control (SPC) as a management technique for software development, software organizations are attempting to employ the method for quality and project control. The focus of these efforts has primarily been with organizations having a Software Engineering Institute Capability Maturity Model® (CMM®) Level 4 or 5 rating. A few CMM Level 4 and 5 organizations have experimented with applying SPC control charts to another management technique – Earned Value Management (EVM). This article discusses the application of SPC control charts to the EVM indicators, schedule, and cost performance indexes.

The software division at the Oklahoma City Air Logistics Center was assessed as Software Engineering Institute (SEI) Capability Maturity Model® (CMM®) Level 4 in 1996, and became registered under the ISO 9001 standard for Quality Systems in 1998. The ISO registration was under the software implementation of the ISO standard known as "TickIT." For these accomplishments and several others, the software division was the recipient of the Institute of Electrical and Electronics Engineers' Software Process Achievement Award for 1999, a truly significant award for the division's efforts.

A large portion of the division's success has been due to embracing the Earned Value Management (EVM) methodology. EVM provided the needed structure to achieve many of the CMM Level 2 and 3 Key Process Areas (KPA) of the SEI's CMM. And, due to its numerical basis, EVM facilitated the achievement of the CMM Level 4 KPA, Quantitative Process Management (QPM), at that time.

However, today the updated QPM KPA strongly urges using control charts for statistical process control (SPC) with the new goal: "Statistically Manage the Sub-Processes [1]." CMM evaluators are now looking for SPC control charts as evidence of satisfying this KPA. Along with the rest of the software industry, we have struggled to meaningfully apply SPC control charts.

Although there is growing evidence of organizations following the CMM goal by implementing SPC with the defect data obtained from peer reviews, only a handful of organizations are employing the technique for controlling and improving software development process performance. The performance application is more difficult, but we believe it has more far-reaching results [2, 3].

Furthermore, we believe the application to performance management is more in line with the intent of SPC, i.e., SPC is intended to optimize the performance of a system, not a component subsystem. The quality guru of the 1980s, Dr. Edward Deming, warned against applying SPC to sub-processes by themselves; he believed these actions could lead to optimizing the sub-process, possibly at the expense of the system. Thus, the following discussion concerns the application of SPC to managing the performance of software projects.

> *"Statistical process control charts, if successfully applied, can be a significant impetus for software process improvements."*

Statistical Process Control

There are several methods for performing SPC: scatter diagrams, run charts, cause and effect diagrams, histograms, bar charts, Pareto charts, and control charts [4, 5]. Although all of these methods are useful, we will focus this article on control charts.

SPC control charts, if successfully applied, can be a significant impetus for software process improvement. The method provides distinction between normal and anomalous process data; it is, in effect, a filter [6]. By knowing our normal process, we can reengineer it to obtain improvement in some performance aspect. And, by identifying anomalous behavior, we can seek the special cause (an influence from outside the system) and take action to prevent it from affecting future performance.

The fundamental idea of process improvement is that as the system is observed over time, the process decreases its variation and, increasingly, gets closer to achieving its planned performance objective because of the introduction of improvements. SPC control charts facilitate this process improvement concept. Thus, you have the reason why the recently issued Software CMM Integration℠ (CMMI℠) [1] has specifically used the words "statistically manage" in its CMM Level 4 Process Area, "Quantitative Project Management."

There are seven SPC control chart types, each having a specific application [4, 5]. The control chart required for our application is termed "Individuals and Moving Range." Symbolically, it is shown as XmR, where X represents the individual observations, and mR represents the moving range, the difference between successive observations. The XmR control chart is used when there is only one measurement of the variable in an observation period.

For all types of control charts, the control limits establish the filtering mentioned earlier. The high limit is plus three sigma from the average of the observations, whereas the low limit is the average minus three sigma. Sigma is a standard statistical measure of the variation in the process. An estimate of sigma is determined from the moving range. Measured values outside of the control limits have an extremely low probability of occurrence, only 0.27 percent if the process follows a normal distribution. Thus, any measured value beyond the control limits is deemed an anomaly, or in SPC terminology, a "signal," and should be investigated by management.

SPC is a much more involved subject than has been discussed here. Significantly more complete information is available in the references [4, 5, 6].

℠ CMMI and CMM Integration are service marks of Carnegie Mellon University.

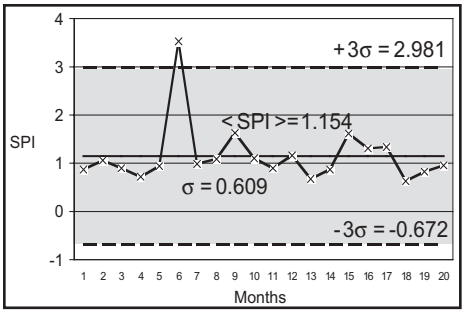


Figure 1: *SPI Control Chart*

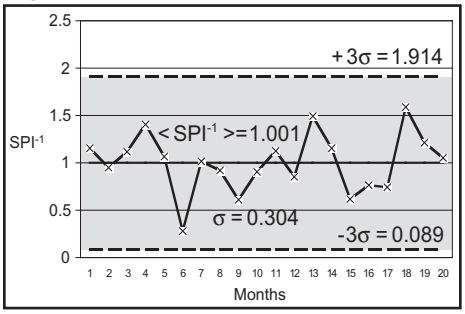


Figure 2: *SPI-1 Control Chart*

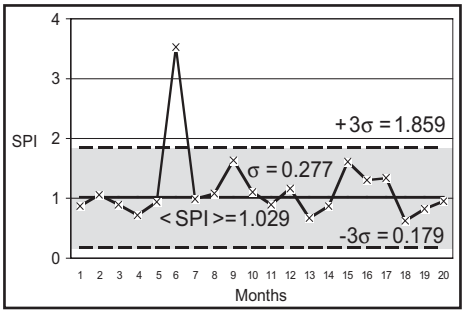


Figure 3: *SPI Control Chart (n=6 Removed)*

Earned Value Management

An excellent reference for EVM is a book by Quentin Fleming, *Cost/Schedule Control Systems Criteria, The Management Guide to C/SCSC* [7]. Just as with SPC, EVM is much more involved than the discussion in this paper. Here, we will only introduce the EVM indicators "cost performance index" (CPI) and "schedule performance index" (SPI).

EVM is based upon establishing a project baseline to achieve the "budget at completion" (BAC); BAC identifies the cost and completion points for the project manager. The baseline performance is a S-curve termed Budgeted Cost for Work Scheduled (BCWS); it is a graph of expected cost versus time. The in-process performance tracking is facilitated by two other curves, Actual Cost for Work Performed (ACWP) and Budgeted Cost for Work Performed (BCWP). BCWP is the earned value; it represents the completion of project tasks and is traceable to the values of cost and time duration allocated to those tasks during the project planning.

During project execution, the CPI and SPI indexes provide information about performance efficiency. The indexes are ratios. SPI is the efficiency of achieving earned value with respect to the performance baseline (SPI=BCWP/BCWS). Similarly, CPI is the efficiency of achieving earned value with respect to the actual costs (CPI=BCWP/ACWP).

Application/Data Analysis

Approximately three years ago, we began applying SPC to the EVM indicators SPI and CPI. We believed the merging of the two powerful management techniques held a considerable amount of promise. Our concept was that the application of SPC control charts to the monthly SPI and CPI values could be used in the following ways:

1. As a predictor of performance for the remainder of a project in work.
2. To improve the planning of new projects by using historical data from completed projects.
3. To effect process improvement, i.e., improve both execution and planning by using the measures of variation (sigma) in monthly performance and variance from the project plan (planned cost and completion date).

As stated, we have been using the method for some time. We have shared the ideas and results in two previous articles [2, 3]. Our results thus far indicate the method will fulfill its promise. However, its employment does require some additional understanding.

When we began preparing the control charts, we observed that the representation of the data affected the analysis and calculated results. To illustrate, we will use a small sample of actual data represented as both SPI and inverse SPI. Control charts for each data representation are shown in Figures 1 and 2. For the SPI chart, a signal is indicated at data point six. By removing the statistically anomalous data point six, the true process performance can be obtained. The control chart for SPI with data point six omitted from the calculations is shown in Figure 3. The true process has an average value of SPI (symbolically, <SPI>) and an estimate of sigma (σ) equal to 1.029 and 0.277, respectively. The inverse SPI chart (Figure 2), however, indicates there are no signals. Therefore, the true process for this data representation has an average value of SPI^{-1} equal to 1.001, while sigma is estimated to be 0.304. As you can clearly see, the analysis results for SPI and SPI^{-1} are not equivalent.

Problem/Proposed Solution

Of course, we should not expect the average values to be equal for the SPI and SPI^{-1} analysis. However, if the signals found and the estimates of sigma are not identical for the two data representations, then we must ask the question, "Which result is correct, or is neither?" If we do not have a basis for choosing a way to represent the data and perform the analysis, then none of the three desired outcomes expressed in the Application/Data Analysis section are achievable.

Another problem can be seen from the histograms of CPI and CPI^{-1} shown in Figure 4. The histograms were created from nearly six years of monthly data from one of our software development projects. By visual inspection, these histograms indicate that the data distributions are probably not "normal." Thus, predictions made by applying a normal distribution to the population would likely be inaccurate [5]. Therefore, similarly to the discussion in the preceding paragraph, unless there is a way to correct the behavior of the data, we cannot use the SPC information derived from the CPI and SPI data for the performance prediction, project planning, and process improvement applications cited earlier.

There are several recognized correction methods that can be used when the distribution of the data is not normal [5]. However, the most appealing is to transform the data in a mathematical way to approximate a normal curve. This is the solution approach discussed in the remainder of the article.

As we became more curious about the differences in the results from the control charts of SPI versus SPI^{-1} and CPI versus CPI^{-1}, we noticed a general bias. The average of the monthly values for either representation is generally larger than its corresponding cumulative value (e.g., <SPI> > SPI_{cum}) and the signals found using XmR control charts are predominantly the observations having values greater than 1.0. Our analysis indicates the problem occurs because the performance indicator (PI) values below 1.0 cannot be less than zero. It is impossible to have a negative value for the PI because it is, simply, a ratio of two positive numbers. However, the values of the PI above 1.0 are unlimited[1].

This behavior of the PI was deduced to be incongruent with the three sigma process limits computed for the individual control chart. The process limits themselves are unbounded; conceivably, they can have values ranging from plus to minus infinity. The process limits are equally spaced above and below the PI average value. However, equivalent good and poor observed values for the PI are not spaced equally above and below the nominal value of 1.0. The PI values less

than 1.0 are virtually ignored; observed performance values below the lower process limit are rare occurrences. For example, signals identified as values higher than the upper process limit for non-inverted data are not detected when the data is inverted. Both Figures 1 and 2 illustrate this inconsistency. Data point six is identified as a signal by the SPI control chart (Figure 1). However, when the data are inverted, the control chart for SPI^{-1} (Figure 2) indicates data point six as nothing unusual; it is within the lower process limit, and thus, is considered to be part of the process. Another significant observation from Figure 1 is that the lower process limit value is below zero; therefore, any SPI value less than 1.0 cannot be detected as a signal.

To make this point a little clearer, let us use SPI (BCWP divided by BCWS). Suppose BCWP is five units and BCWS is one unit. Now suppose the performance is reversed; BCWP is one unit, and BCWS is five units. For the first instance, SPI_a=5.0, and for the second, SPI_b=0.2. The two instances are, numerically, the reciprocal of the other and represent equivalent anomalous performance. SPI_a=5.0 is excessively good schedule efficiency, whereas SPI_b=0.2 is excessively poor schedule efficiency. The two values are not equidistant from 1.0, the nominal value; SPI_a is four units away from 1.0, whereas SPI_b has a separation of only 0.8 units.

If we constructed a control chart with these data points included, SPI_a would likely be detected as a signal. And if the lower process limit happened to be a negative number, SPI_b would be seen as part of the process. However, for the inverse data representation the signal detected becomes SPI_b^{-1}=5.0, while SPI_a^{-1}=0.2 is ignored. The signals identified are switched with the change in data representation. Thus, it should now be understood that the SPC control limits generally detect EVM performance index indicator signals greater than 1.0 and ignore poor performance values less than 1.0, regardless of whether the data are represented as inverted or non-inverted.

Due to the incongruence between the PI data and the XmR process limits, a method was sought to transform those PI (or PI^{-1}) values less than 1.0 from being bounded by zero to unbounded. It was hoped that if the data values less than 1.0 could be made to resemble the characteristics of those values greater than one, then consistent results could be obtained from data represented as either inverted or non-inverted.

The method for representing all of the data as unbounded is extremely simple. The PI data are transformed for the SPC analysis by using the natural logarithm function. Applying the natural logarithm function causes PI values less than 1.0 to be represented by negative numbers. The transformed values are a data set that is congruent with its corresponding three sigma process limits. Thus, when the transformed data are used for creating the SPC control chart, a negative value lower process limit will not necessarily mean that a signal whose value is less than 1.0 will be ignored. The transformed data provides the possibility of identifying signals for both high and low PI values.

Solution Criteria/Testing/ Results

With the mathematical method defined for transforming the data, tests can be performed to determine whether or not the application of the logarithm function meets a set of desirable behavior characteristics. Fundamentally, if the solution is the correct one, it should not matter which data representation is used for the SPC analysis, inverted or non-inverted.

Specifically, the un-transformed average value of the inverted data should be the reciprocal of the un-transformed average value of the non-inverted data[2]. The value of sigma representing the process variation for the non-inverted data should equal the value of sigma determined from the inverted data. The same data points should be identified as signals in either data representation. Lastly, the transformed data should show improved agreement to the normal distribution. If the solution meets all of the criteria, we can feel confident in its use.

To test the transformation, it is applied to the SPI data previously analyzed (see Figures 1 and 2). The transformed SPI data, both the non-inverted and inverted representations, are shown in the SPC control charts, Figures 5 and 6, respectively. Reviewing and comparing these control charts, it can be said that the results satisfy three of the four solution criteria:

1. The un-transformed average values, i.e., $<SPI>_u$ and $<SPI^{-1}>_u$, are 0.994 and 1.006, respectively; $<SPI^{-1}>_u$ is the reciprocal of $<SPI>_u$.
2. The signal identified in Figure 5 is identical to the signal found in Figure 6, i.e., data point six. Figure 6 is the mirror image of Figure 5 with respect to the ordinate value 0.0.

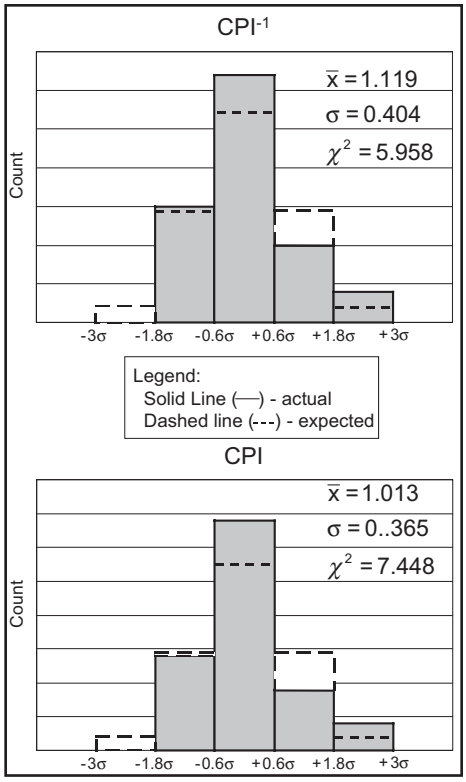


Figure 4: *Histograms – CPI and CPI^{-1}*

3. The estimate of sigma is the same value for both the inverted and non-inverted representations of PI; i.e., the value of σ is 0.259.

One test using a small data sample certainly is not proof that the transformation meets the criteria in every instance. However, we have run the analysis using the natural logarithm transformation on data from several projects (more than 400 data points) and have seen consistent results. Furthermore, although it is beyond the scope of this paper, through mathe-

Figure 5: *ln(SPI) Control Chart*

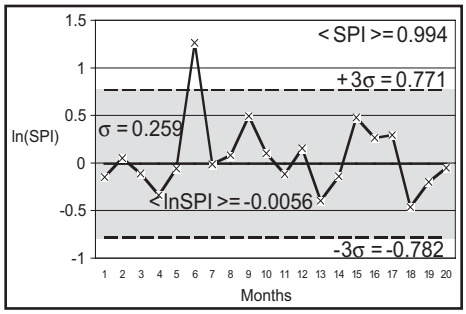
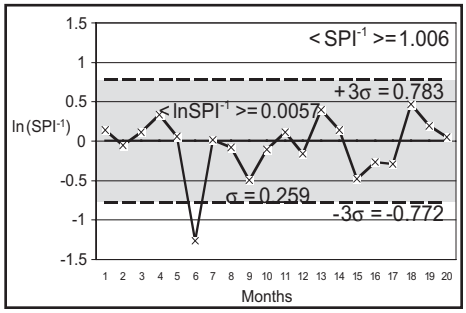


Figure 6: *ln(SPI-1) Control Chart*



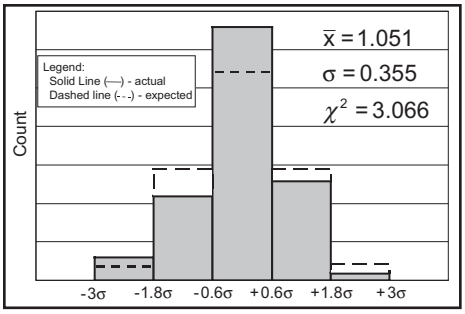


Figure 7: *Histogram – InCPI*[1]

matical analysis it can be shown that each of these three criteria for data representation equivalence is achieved by applying the natural logarithm function.

To illustrate that the transformation improves the normality, visually compare the histogram shown in Figure 7 to its counterpart in Figure 4. The same data are used for both histograms; however, Figure 7 uses the transformed values. The histogram of the logarithm values has a more normal appearance. To further explain the improvement, the "goodness of fit" Chi-Square (χ^2) statistical test [5, 8] is applied to the histograms[3]. Smaller values of Chi-Square indicate more conformity to a normal distribution. The Chi-Square value for the raw CPI^{-1} data is 5.958 (see Figure 4), whereas the value for the transformed data is 3.066 (see Figure 7). Thus, there is 48.5 percent improvement in Chi-Square by the use of the natural logarithm function.

In addition to high and low signals being detected equally, another significant outcome from using the natural logarithms of the data is that the resulting average value of the performance indicator ($<PI>_u$) is unbiased. It is not generally greater than the cumulative value of PI (PI_{cum}). Thus, the average performance indicator values, $<SPI>_u$ and $<CPI>_u$, and their reciprocals can be used with confidence in EVM calculations.

Summary

SPC applied to the EVM indicators, CPI and SPI indexes, can be a very powerful management tool. It has immense potential to improve both the planning and execution of software development projects and to provide a measure of that improvement. We have shown, however, that the application has inconsistencies that detract from its employment. A solution is proposed for resolving the problem. A test for the solution is described and performed using a small set of SPI data. The results of more extensive testing and further mathematical analysis indicate the recommended solution has merit. Employing the data transform technique significantly reduces the SPC-EVM application inconsistencies, thereby yielding much improved results.◆

References

1. CMMI-SE/SW, Version 1.02, CMU/SEI-TR-018, CMMI Product Development Team, Nov. 2000.
2. Lipke, Walt, and Jeff Vaughn, eds. "Statistical Process Control Meets Earned Value." CROSSTALK June 2000: 16-20.
3. Lipke, Walt, and Mike Jennings, eds. "Software Project Planning, Statistics, and Earned Value." CROSSTALK Dec. 2000: 10-14.
4. Florac, William A., and Anita D. Carleton, eds. Measuring the Software Process. Reading, Mass.: Addison-Wesley, 1999.
5. Pitt, Hy. SPC for the Rest of Us. Reading, Mass.: Addison-Wesley, 1995.
6. Wheeler, Donald J. Understanding Variation, The Key to Managing Chaos. Knoxville, Tenn.: SPC Press, 2000.
7. Fleming, Quentin. Cost/Schedule Control Systems Criteria, The Management Guide to C/SCSC. Chicago: Probus, 1988.
8. Crow, Edwin L., Francis A. Davis, and Margaret W. Maxfield, eds. Statistics Manual. New York: Dover Publications, 1960.

Notes

1. The observations discussed in this paragraph are made for the non-inverted data representation. The statements apply equally as well for the inverted data representation (PI^{-1}).
2. The reciprocal relationship of the untransformed average values is also needed for EVM calculations. The cumulative values of the EVM performance indicators, represented as inverted and non-inverted data, possess this characteristic, i.e., $(PI_{cum})^{-1} = (PI^{-1})_{cum}$. Thus, to be confident using the numbers from the SPC analysis in EVM calculations, they must behave in accordance with the cumulative values.
3. The definition of Chi-Square for this application is: $\chi^2 = \Sigma_i [(\text{expected count}_i - \text{observed count}_i)^2 / \text{expected count}_i]$, where "i" designates one of the five histogram areas (e.g., 0.6σ to 1.8σ).

About the Author

Walt Lipke is the deputy chief of the software division at the Oklahoma City Air Logistics Center. He has 30 years of experience in the development, maintenance, and management of software for avionics automated testing. 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 the Software Engineering Institute's Capability Maturity Model® (CMM®) Level 2. 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. Lipke is a professional engineer with a master's degree in physics.

OC-ALC/LAS
Tinker AFB, OK 73145-9144
Phone: (405) 736-3341
Fax: (405) 736-3345
E-mail: walter.lipke@tinker.af.mil

Are You Prepared for CMMI?

Suzanne Garcia
Software Engineering Institute

For those making the transition to the Capability Maturity Model® Integration℠ (CMMI℠) from another process improvement model or methodology, understanding the transition as a technology adoption and applying technology adoption concepts can smooth the process considerably. In this article, technology adoption concepts are described and then exemplified in a CMMI context. Some of these are already well known within the software process improvement industry, others are not.

This article focuses on applying technology adoption concepts in moving toward using the Software Engineering Institute's (SEI) Capability Maturity Model® (CMM®) Integration℠ (CMMI℠) framework. It is assumed that the reader has a basic understanding of the CMMI concepts and the project that formulated it. If not, please see the CMMI area within the SEI's Web site at <www.sei.cmu.edu>. For more in-depth information, see CMMI Distilled [1] for basic information on the model and the project written by some of the CMMI project team members.

CMMI Adoption as Technology Adoption

What is technology adoption? Generally, it is the set of practices and factors related to organizations selecting, deploying, and sustaining the use of a technology. Why look at CMMI adoption as "technology" adoption? First, CMMI "is" a technology (a tool or tool system by which we transform parts of our environment, derived from human knowledge, to be used for human purposes [2]) – a "process technology" – and what is more, it is "radical."

"Radical innovation is the process of introducing something that is new to the organization and that requires the development of completely new routines, usually with modifications in the normative beliefs and value systems of organization members [3]." Treating CMMI as a technology adoption first mobilizes a different mindset than the one we typically apply to process improvement, and second may make us more inclined to use some of the useful tools of technology adoption for our CMMI adoption.

Technology Adoption Concepts

Given that the factors involved in technology adoptions are complex, it stands to reason that each adoption is highly situational; its strategy will be unique to that situation and context. Some basic concepts can, however, be applied in generating that unique strategy, including the following:

1. Multiple dimensions have to be addressed simultaneously to achieve success, not just the technology (in this case, CMMI) content.
2. Different audiences with different roles and responsibilities in an organization respond differently as they are introduced to the technology.

> *"Acceptance of a new technology does not happen in a linear, predictable fashion no matter how pretty the charts look!"*

3. Acceptance of a new technology does not happen in a linear, predictable fashion no matter how pretty the charts look!
4. There are both different "levels of diffusion" (breadth of technology acceptance) and "levels of use" (degree to which the technology becomes embedded in the organization's governing and social practices). One does not imply the other.
5. Different "mechanisms" are useful at different points in the transition to address different implementation issues with different audiences.
6. Most organizations are very poor at transferring what they've learned from one technology adoption effort to another. Communities of practice are one strategy for addressing this.

The rest of the article will focus in turn on each of these dimensions.

Dealing With Multiple Dimensions Simultaneously

In talking to individuals and groups contemplating CMMI adoption, I have sometimes run into this mindset: "We've been successful at implementing Software CMM (SW-CMM); what's the big deal about implementing CMMI?"

From one viewpoint, this is an attractive mindset – it indicates that the individuals speaking see strong similarities between SW-CMM version 1.1 and the CMMI framework. Indeed there are many similarities, although there are more between SW-CMM version 2 draft C and CMMI since version 2 draft C was one of the seed documents for CMMI, rather than version 1.1.

However, the CMMI framework also provides an opportunity to expand the scope of application of CMM concepts beyond just the software organization into the other parts of the organization involved in product or service development. This means involving new players in the CMM adoption and expanding the scope of effect of CMMs on the subsystems of the organization.

Consider this: Which subsystem elements in Figure 1(see page 20) are "the same" in context, roles, or resources between different disciplines such as software engineering and systems engineering in your organization? Creating alignment among these elements is not a sequential process. For example, changing the technical practices of the organization could have a strategic effect if those practice changes enable the organization to compete in a marketplace that was previously closed to them.

The managerial and structural subsystems almost always have to be dealt with in parallel. The social/cultural subsystem provides an underpinning for all the other subsystems, and the negative effects of neglecting the social design when redesigning other elements is well-established [1]. However, if these subsys-

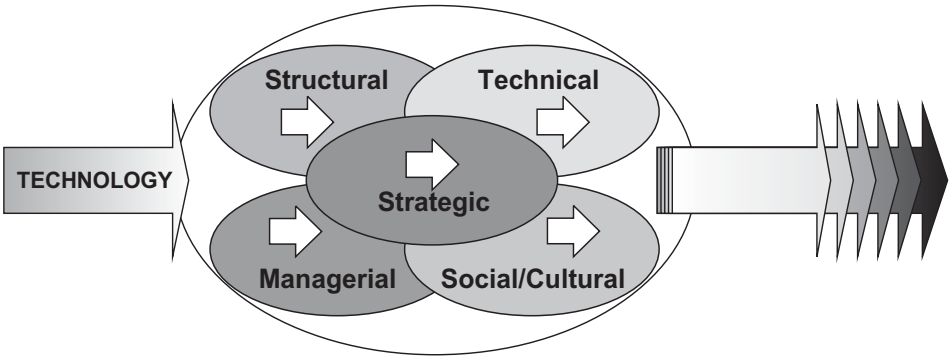


Figure 1: *Organizational Subsystems*

tem elements are aligned in the same direction, perhaps via use of a similar improvement model like CMMI, then not only is technology adoption smoother, but operations are also often smoothed as well. The elements that follow, in many cases, cross more than one of these subsystem elements.

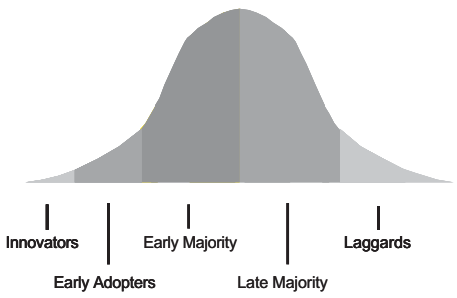
Understanding Your Audience

Who in your organization has to change something in their behavior, attitudes, or values to adopt CMMI: executives, managers, technology users, support groups? What things make these groups more or less likely to change? Edgar Schein's work in organizational subcultures cites distinct differences, for example, between the executive and engineering cultures.

"Executives and engineers are task focused and assume that people are the problem. Executives band together and depersonalize their employees. Executives and engineers can't agree on how to make organizations work better while keeping costs down. Enough mutual understanding must be created among the cultures to evolve solutions that all groups can commit to [3]."

Within subculture groups, "individuals" also differ in their response to a technology adoption. Different "adopter types" move through adoption at different speeds. These groups are distinguished from each other by their characteristic response to an innovation (either process or technology) that requires a change in their behavior. Figure 2 illustrates where each adopter category falls in the technology adoption life cycle. Geoffrey Moore's *Crossing the Chasm* [4] and *Inside the Tornado* [5] popularized the description and use of these characteristics in great detail, based on ongoing work in the diffusion of innovations research area by Everett Rogers [6]. A brief summary of each adopter type is provided in the sidebar for those who are not familiar with this work.

Figure 2: *The Technology Adoption Life Cycle*



Adoption populations are used extensively in planning technology adoption strategies. Understanding the adoption category of intended early users is extremely important. For example, if an intended pilot for new practices turns out to be a group composed primarily of late majority or laggard participants in relation to that set of practices, then I could easily predict the following: Any adoption that (1) does not provide a completely packaged solution and (2) is not mandated by the organization, including sanctions for not adopting, is highly likely to fail. In addition to planning who will get the technology when, adopter categories can be used to categorize what kinds of adoption support mechanisms (see "Transition Mechanisms" section) are likely to be needed to ensure that each category of interest is more likely to successfully adopt the technology.

Adoption: Not a Linear Process

Work done in the educational innovations area has provided valuable insight in understanding the patterns that are often operating as a technology adoption such as CMMI adoption occurs. The original chart used by Patterson and Conner [7], reproduced as Figure 3, shows several milestones of increasing commitment to an adoption as time progresses.

The SEI has found that the path through these milestones is rarely linear, and there are a plethora of approaches for successfully navigating this progression, depending on the individual situation. However, a couple of points in Figure 3 are worthy of note:

- There are clear signs if the organization has "dropped out" of the adoption process. Looking at the behaviors of the organization in relation to the technology can provide a clue as to the point in adoption where translation to the next milestone was not made successfully.
- "Understanding" – the point at which decision makers in the organization have sufficient knowledge and context information to make a relevant decision about the technology – does not typically happen on first, or even second contact. This has always helped me to be tolerant of organizational members who "aren't getting it." I ask myself, "Have I given them enough contact and depth to have the right to 'expect' understanding?"

Diffusion vs. Infusion

In considering a particular process technology adoption like the CMMI, some time should be spent determining the adoption goals. Some of the other concepts previously described such as what kinds of adopters the adoption is targeting, what elements of the organization need to be realigned, etc., can be used to help set some of these goals.

Another area related to setting goals that should be considered is the relative emphasis that will be placed on CMMI "diffusion" (how widespread the use of CMMI has become) vs. CMMI "infusion" (how deeply embedded into the organizational infrastructure the CMMI has become). This latter area, technology infusion, focuses on the extent to which the work system and social system of the organization are affected by the technology. To measure infusion, one can measure "levels of use" of a technology. For example, the evolution of the infusion of CMMI use in an organization might look something like this:

1. CMMI adoption has occurred in a few projects whose local procedures and processes have been changed to reflect the new practices.
2. One of the divisions of the organization has changed its policies to reflect the practices recommended in CMMI and has formulated and published a set of standard process assets that are used as the basis for initiating and managing new product development

projects.

3. Reward and incentive systems in the new projects adopting CMMI practices have been examined and changed where necessary to encourage productive use of the new processes. Existing projects within the division have been evaluated to determine which parts of the set of standard process assets might beneficially be applied to the projects at their current point in the life cycle. Projects are being provided the training and other support needed to make it feasible for them to adopt new practices in mid-project.
4. Members of projects in the division adopting the CMMI are being recruited for projects in other parts of the organization due to the projects' reputation for meeting customer expectations. However, many of them choose to stay within the division rather than move to the other parts of the organization that are less disciplined in their management and engineering practices.

Each of these scenarios could be considered a level-of-use measure for the infusion of CMMI adoption within the organization. With increasing levels of use, the degree of workflow interconnectedness related to the CMMI use increases, and the degree of visibility of the technology within the social subsystem is increased, as exemplified in the example of the fourth scenario.

Zmud and Apple, the authors of various articles in this area [8], recommend that an organization that wants to understand the infusion of a technology into the organization identify different configurations that reflect the levels of use that are the goals for the adoption as time continues. Like the CMMI itself, this is a cumulative approach since each configuration builds on the prior configuration's functionality.

This viewpoint has particular applicability to a technology like the CMMI, which in a sense has several embedded configurations enabled by the capability levels or maturity levels. For those using the staged view of CMMI, the maturity levels provide a priori configurations that translate to certain functionality related to the model being present. For those organizations for whom the staged-view functionality is not compatible with their business drivers, thinking of groups of process areas to be adopted as a "configuration" for measuring levels of use is one way to provide measurable anchor points to the improvement effort.

Adopter Types

- Innovators are gatekeepers for any new technology. They appreciate technology for its own sake from its architecture to its application. They will spend hours trying to get technology to work and are very forgiving of poor documentation, slow performance, and incomplete functionality. Innovators are helpful critics to technology producers who are willing to fine-tune their products.
- Early adopters are dominated by a dream or vision, focusing on business goals. They usually have close ties with "techie" innovators, so as to be ready to match emerging technologies to strategic opportunities. They thrive on high visibility, high-risk projects, and have the charisma to generate buy-in for them. They do not, however, have credibility with early majority types, so they have limited influence over that group.
- Early majority adopters do not want to be pioneers (prudent souls). They control the majority of the budget, so they want to be assured of percentage improvement (incremental, measurable, predictable progress). They are not risk averse, but do want to manage change carefully. They are hard to win over, but are loyal once won and will often be the people who will carry a new initiative through into institutionalization.
- Late majority adopters avoid discontinuous change (change that requires them to change their behavior) whenever possible: They adopt only to stay on par with the rest of the world. Somewhat fearful of new technologies, they prefer pre-assembled packages with everything bundled. In comparison with early adopters or early majority populations, they are much less willing to invest in a technology that requires visible change to their practices, unless mandated.
- Laggards are the "naysayers" in the crowd. They adopt only after the technology is not recognizable as a separate entity, after it has become part of "the way things are done around here." They do, however, constantly point at discrepancies between what was promised and what is delivered. When contemplating a technology adoption, they are often very useful for identifying risks (both technology-based and culturally-based) to the adoption.

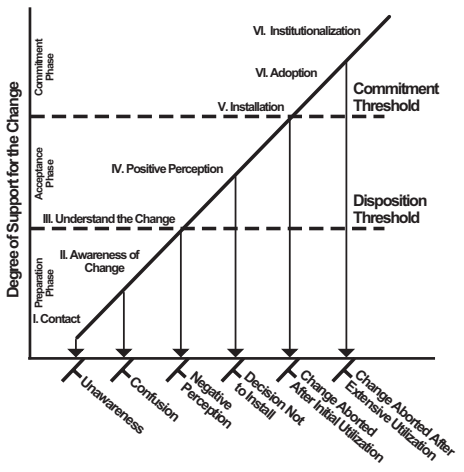
Diffusion – how broadly the technology has penetrated – is also an important measure. One of the most useful ways of approaching this that I have encountered is found in Kim Caputo's book *CMM Implementation Guide: Choreographing Software Process Improvement* [9]. She suggests that the organization "operationalize" the stages of the Patterson-Conner commitment curve ("What does it mean for us to achieve awareness, understanding, etc.?") in terms of events and symptoms of behavior change; then use a histogram to show the organization's population against those events and behaviors.

At the beginning, one might expect that the organization might have a profile like Figure 4 (see page 22). As time goes on, the profile should shift to something like that shown in Figure 5 (see page 22) as more and more members of the organization participate in the activities of CMMI adoption. One use of this measure is to help senior managers understand the time needed to see tangible return on investment of a CMMI implementation. When they understand how many people have to go through several events before one can expect their behavior, and therefore their results, to change, it can help them tolerate some of the time lag that is typical between starting an adoption effort and seeing business results.

Transition Mechanisms

Transition mechanisms are products, events, and methods for translating from one commitment milestone to another.

Figure 3: *Patterson-Conner Commitment Curve*



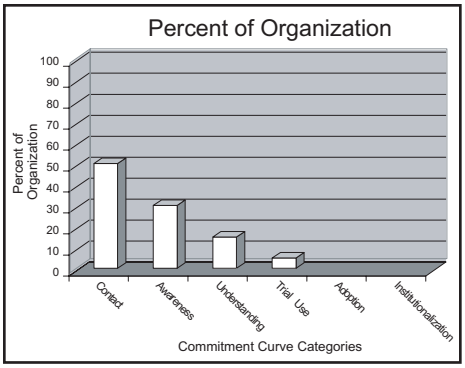


Figure 4: *Notional Profile Early in Adoption*

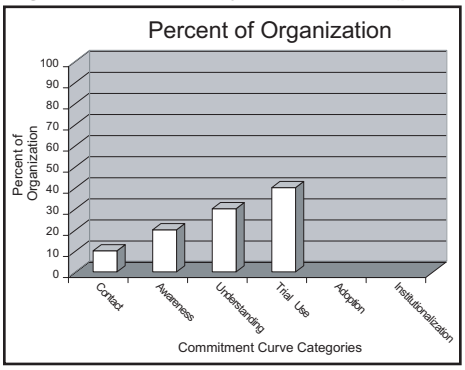


Figure 5: *Notional Profile Later in Adoption*

Many transition mechanisms are typically developed by the technology provider. By translation, we mean the process of communicating about the CMMI adoption in terms and language that are likely to be understood and usable by the individuals/groups we are working with. As we proceed in the adoption, the mechanisms move in character from communication and education more toward implementation support and incentives management.

See the "Adoption Mechanisms" sidebar for an example set of mechanisms that could be used in your organization for each CMMI adoption stage. Which ones are right for you depends on your organization's context and culture, and the list is certainly not exhaustive. However, it is based on the list elicited from the May 2001 "The Road to CMMI" technology transition workshop hosted by SEI's Accelerating Software Technology Adoption (ASTA) for organizations that are already using CMMI to support their improvement efforts.

A resource that is available to the community to help understand how early adopters of CMMI have approached their adoption is a presentation of "The Road to CMMI: What Works, What's Needed" [10] from the CMMI Technology Conference and User Group held in Denver in November 2001. Also keep an eye on the SEI publications page for the full technical report that will be published from the workshop results. The reader can access the SEI publication page at <www.sei.cmu.edu/publications>.

Integrating Transition Mechanisms and Commitment Milestones

Figure 6 shows an SEI adaptation of the Patterson-Conner commitment curve with commonly used adoption mechanisms embedded with each stage. Note that rather than a straight line, it shows a more curved progress. This is intended to reflect the observation that organizational investment increases significantly once "understanding" has been achieved and the organization is trying to achieve "trial use" and then "adoption."

Innovators and Early Adopters will tend to create their own transition mechanisms and make do with what is available from the technology producer. Early and Late Majority adopters expect many of these mechanisms to be readily available for them to acquire without development.

Technology producers (in this case, the CMMI Product Team) and SEI transition partners [11] often have many of the mechanisms in "contact, awareness, and understanding" available in their marketing kits. Technology adopters usually have to adapt these to help "sell" the technology to the intended users. Transition mechanisms can include events and activities as well as "products."

Building a Community of Practice

One of the exciting areas of research at the SEI within its ASTA initiative is exploratory work that is being done in seeding and helping to sustain "communities of practice." This concept is one of the underpinnings of much of the work in knowledge management, but is particularly useful when looking at adopting a technology like the CMMI. The SEI is incorporating this concept into a larger research project called KNiTT (Knowledge Networks in Technology Transition), which looks at concepts from the communities of practice literature as well as systems engineering techniques, case-based learning, and knowledge repositories to formulate environments and approaches to supporting a technology adoption context.

Even before this work matures, some of the early ideas are worth considering in your CMMI adoption. One of the crucial ideas in this arena is the notion that deep learning about a new technology tends to be problem-centric, that is, the technology will be evaluated by potential users when there is a problem they are trying to solve that appears to be a match

Adoption Mechanisms From the Road to CMMI Workshop

Contact and Awareness

- Identification of Capability Maturity Model® Integration℠ (CMMI℠) communication channels.
- CMMI awareness briefings and forums.
- Organization's history and context.
- Translations of SEI materials into the organization's language/context.
- CMMI reference cards, other promotional materials.

Trial Use

- Integrating quality assurance (QA) to measure process improvement (PI) progress.
- Link QA process to CMMI.
- Transition strategy from SW-CMM to CMMI.
- Pilot/trials in non-software-development areas.
- Example CMMI PI budget.

Institutionalization

- CMMI best practice-based templates/checklists/assets.
- Integrating process review into project management review.

Understanding

- CMMI communications repository.
- Self-assessment/gap analysis/class B and C assessments that relate gaps to the organization's processes.
- CMMI poster.
- Process improvement information database (use of data).
- Mapping of various models to CMMI.
- Chart on how processes are the responsibility of different roles/across organizational boundaries.
- Think CMMI program.
- Transition roadmap.
- Birds-of-a-feather sessions on focused topics.

Adoption

- Role-based training.
- Tailoring guidance/strategies for different organizational contexts.
- Transition steering group.
- Return-on-investment trend data.
- Integrating all disciplines into the process group.

for the technology. Until that time, many of the mechanisms that could be used to support adoption of the technology will help build knowledge and positive impressions of the technology, but will not be able to achieve trial use or adoption.

Once problems are being solved with a technology, the possibility exists to seed a community of practice, which contains members of the organization who are motivated to continue learning about the technology. They might build "translations" of the technology for other users who may not be as far along in their adoption of the technology, and communicate and problem-solve with each other to improve their use of the technology. At this point, a community of practice may be considered to be initiated.

In CMM adoption history, many of the Software Process Improvement Networks (SPINs) exhibit characteristics of communities of practice. We believe that bringing the ideas of continued learning and involvement by the practitioners and change agents inside the organization can accelerate the adoption of a technology like the CMMI, since this approach tends to access the informal networks of influence that exist within the organization outside the normal organizational structure. Stay tuned to the ASTA section of the SEI Web site for information on these and other ASTA research areas as they evolve.

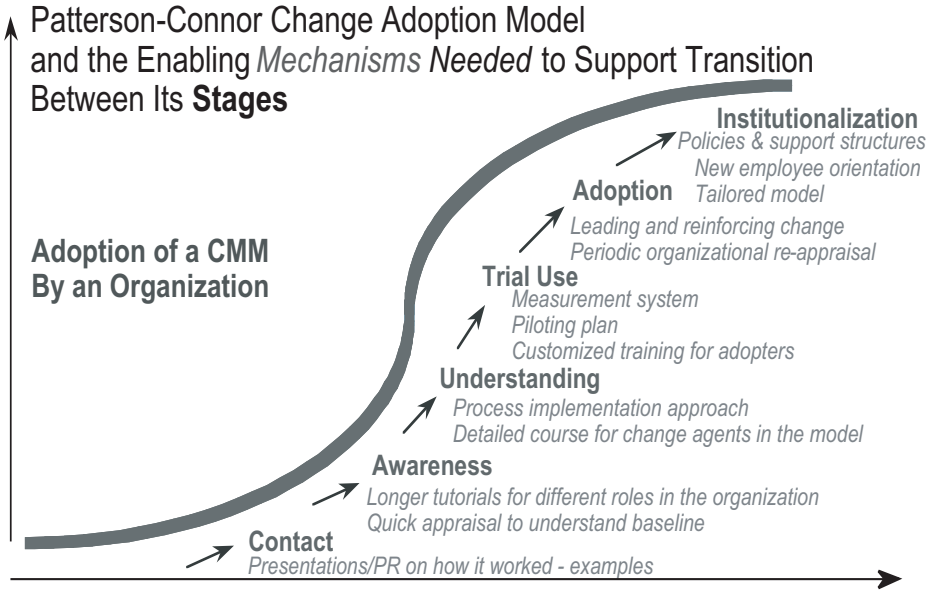


Figure 6: *Notional Commitment Curve for Adopting CMMI, with Possible Transition Mechanisms*

Summary

The CMMI is early in its own maturation/transition life cycle. This means that there are little hard data on successful (and unsuccessful) strategies for its use, and there are no return-on-investment data that a chief financial officer would find credible. There are many approaches from the technology adoption arena that can be useful in making effective use of the CMMI as it matures. A few of these have been highlighted in this article. As an early adopter of CMMI, you need to be prepared to invest in creating the transition mechanisms your organization will need to be successful and to apply creative approaches to making progress. Understanding and applying technology adoption concepts can help you maximize your return on your investment.◆

References

1. Tornatzy, Louis G., and Mitchell Fleischer, eds. The Process of Technological Innovation. Lexington, Mass.: Lexington Books, 1990.
2. Nord, W. R., and S. Tucker, eds. Implementing Routine and Radical Innovations. Lexington, Mass.: Lexington Books, 1987, p. 41, as quoted in [1].
3. Schein, Edgar. "The Three Cultures of Management: Implications for Organizational Learning." Sloan Management Review 38: 9-20.
4. Moore, Geoffrey A. Crossing the Chasm. New York: HarperCollins Publishers, 1991.
5. Moore, Geoffrey A. Inside the Tornado: Marketing Strategies from Silicon Valley's Cutting Edge. New York: HarperCollins Publishers, 1995.
6. Rogers, E. M. Diffusion of Innovations. New York: Free Press, 1995.
7. Patterson, Robert W., and Darryl R. Conner, eds. "Building Commitment to Organizational Change." Training and Development Journal Apr. 1982: 18-30.
8. Zmud, Robert W,. and L. Eugene Apple, eds. "Measuring Technology Incorporation/Infusion." Journal of Product Innovation Management 1992: 9: 148-155.
9. Caputo, Kim. CMM Implementation Guide: Choreographing Software Process Improvement. Reading, Mass.: Addison-Wesley, 1998.
10. Wemyss, Gian. "Results from the Road to CMMI: What Works, What's Needed." Proceedings of the 1st CMMI Technology Conference and User Group, Nov. 2001.
11. Software Engineering Institute. "Transition Partner Program." <www.sei.cmu.edu/collaborating/partners/trans.partners.html>.

Additional Information

1. See the CMMI Web site <www.sei.cmu.edu/cmmi>, or the ASTA Web site <www.sei.cmu.edu/asta>.
2. Contact Software Engineering Institute Customer Relations: Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890; Fax: (412) 268-5800; Web site: <customer-relations@sei.cmu.edu>.

About the Author

Suzanne Garcia is a returning senior member of the technical staff at the Software Engineering Institute (SEI) of Carnegie Mellon University, working in the Accelerating Software Technology Adoption initiative. From November 1997 to May 2001, she worked with aimware, Inc.'s U.S. customers, focusing on technology-supported organizational improvement. She spent the previous five years at the SEI working in various capacities in the Capability Maturity Models initiative. Twelve years prior to that Garcia worked in multiple improvement-related roles at Lockheed Missile and Space Co.

Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213
Phone: (412) 268-9143
Fax: (412) 268-5758
E-mail: smg@sei.cmu.edu

Correctness by Construction: Better Can Also Be Cheaper

Peter Amey
Praxis Critical Systems

For safety and mission critical systems, verification and validation activities frequently dominate development costs, accounting for as much as 80 percent in some cases [1]. *There is now compelling evidence that development methods that focus on bug prevention rather than bug detection can both raise quality and save time and money. A recent, large avionics project reported a four-fold productivity and 10-fold quality improvement by adopting such methods* [2]. *A key ingredient of correctness by construction is the use of unambiguous programming languages that allow rigorous analysis very early in the development process.*

In December 1999 CROSSTALK [3], David Cook provided a well-reasoned historical analysis of programming language development and considered the role languages play in the software development process. The article was valuable because it showed that programming language developments are not *sufficient* to ensure success; however, it would be dangerous to conclude from this that they are not *necessary* for success. Cook rightly identifies other issues such as requirements capture, specifications, and verification and validation (V&V) that need to be addressed.

Perhaps we need to look at programming languages not just in terms of their ability to code some particular design but in the influence the language has on some of these other vital aspects of the development process. The key notion is that of the benefit of a *precise* language or language subset. If the term *subset* has set anyone thinking "oh no, not another coding standard," then read on, the topic is much more interesting and useful than that!

Language Issues

Programming languages have evolved in three main ways. First came improvements in *structure*; then attempts at improving compile-time error detection through such things as *strong typing*; and, most significantly, facilities to improve our ability to express *abstractions*. All of these have shaped the way we think about problem solving.

However, programming languages have not evolved in their precision of expression. In fact, they may have actually gotten worse since the meaning of a sample of machine code is exact and unequivocal, whereas the meaning of the constructs of typical modern high-order languages are substantially less certain. The evolution of C into C++ certainly improved its ability to express design abstractions but, if anything, the predictability of the compiled code decreased. These ambiguities arise either from deficiencies in the original language definition or from implementation freedoms given to the compiler writer for ease of implementation or efficiency reasons.

> *"Programming languages have not evolved in their 'precision' of expression. In fact, they may have actually gotten worse ..."*

None of this may look like a very serious problem. We can still do code walkthroughs and reviews and, after all, we still have to do dynamic testing that should flush out any remaining ambiguities. In fact the evidence is quite strong that it *does* matter because it creates an environment where we are encouraged to make little attempt to reason about the software we are producing at each stage of its development. Since we typically do not have formal mathematical specifications and we use imprecise programming languages, the first artifact we have with formal semantics is object code (object code is formal in the sense that the execution machine provides operational semantics for it). So our first opportunity to explore the behavior of the software in any rigorous fashion occurs very late in the development cycle with malign consequences.

Where this trend is most harmful is in high-integrity systems where reliability is the pre-eminent property required. The most common form of high-integrity system is the safety-critical system. Such systems are characterized by the proportionally very high overall effort that goes into showing that the system is fit for service: the V&V effort. We are seeking to demonstrate *before there is any service experience* that a system *will be* fit for use.

Claimed failure rates of 10^{-9} per flying hour are not uncommon in aviation: 10^{9} hours is more than 114,000 years! Leaving aside for now the question of whether we can ever hope to demonstrate such levels of integrity by dynamic testing alone [4, 5, 6], what is certain is that any such attempt will be expensive. For high-integrity systems where typically more than half – and sometimes as much as 80 percent – of the time is spent in the integration and validation phases, we are locked into a vicious circle: We are spending most of our time at the most expensive point in the life cycle. Worse, it is the point at which any delay will inevitably affect the overall program schedule.

The consequences of these difficulties are well recognized. It is often stated that developing software to a standard such as DO-178B[1] [7] at level A raises the cost by a factor of five over non-critical developments. Much of this extra cost comes from meeting the specific test coverage criterion of the standard.

Reliable Programming in Standard Languages

If we want to avoid the vicious circle of late error detection and costly repair we must start to reason logically about our software at an earlier stage in its development. We can do this by using a programming language whose source code has a precise meaning; this makes it possible to provide tool support in the form of static analyzers[2] that can be applied very early in the coding process, *before dynamic testing begins*.

This kind of early analysis, at the engineer's terminal, is an amplification of the static analysis (such as type checking) performed by compilers. The aim is to prevent errors ever making it to test. These gains are only possible if the language rules are precise and the semantics are well defined.

Given the imprecision of programming languages in general we have three ways of gaining the properties we seek:

- Work with particular, compiler-defined dialects of programming languages.
- Design new languages with the properties we require.
- Use subsets of mainstream languages designed to have the required properties.

Using dialects is quite a respectable approach but not without its drawbacks. First, vendors may be reluctant to reveal the dialect. Also, there is no guarantee that the compiler's behavior won't change from version to version and without your knowledge. Finally, there is no guarantee that the compiler will even be consistent in its behavior.

Purpose-designed languages with formally defined semantics are a fascinating research topic but are unlikely ever to achieve the kind of critical mass required to make them suitable for widespread industrial use. My favorite example is Newspeak [8]. It is named after the language in George Orwell's novel *1984*. Newspeak is elegant, precise and has all the required properties. It also has several major flaws: you can neither buy a compiler for it, nor a textbook, nor get training, nor hire staff! This is the fundamental drawback of the custom-language approach.

That leaves using coherent subsets of mainstream languages. Done properly this can provide the best of both worlds: precise semantics together with access to tools, training, and staff. However, typically these subsets are informally defined. Some language features may be omitted. There is no attempt to construct a logically coherent sublanguage that makes the qualitative shift to having no ambiguous or insecure behavior. For example, MISRA-C is a C-language subset defined by the United Kingdom (UK) automotive industry. MISRA-C [9] prohibits some of the more problematic constructs of C, such as unrestricted pointer arithmetic, which is frequently the cause of coding problems [10]. However, MISRA-C is not, and does not claim to be, a logically sound sublanguage.

Other subsets such as SPARK[3] [11, 12, 13] are more ambitious since they seek to define a language whose source code wholly defines the eventual compiled behavior. The language is intended to be completely free from ambiguity, compiler-dependent behavior, and other barriers to precise reasoning. Before going on to describe SPARK in more detail it is worth looking further at the practical advantages of correctness-by-construction approach.

The Lockheed C130J

The Lockheed C130J or Hercules II Airlifter was a major updating of one of the world's most long-lived and successful aircraft. The work was done at Lockheed's own risk. Much of the planned aircraft improvement was to come from the completely new avionics fit and the new software that lay at its heart. The project is particularly instructive because it has some unusual properties that provide some interesting comparisons:

"The exact semantics of SPARK require software writers to think carefully and express themselves clearly; any lack of precision is ruthlessly exposed by ... its support tool, the SPARK Examiner."

- Software subsystems developed by a variety of subcontractors using a variety of methods and languages.
- Civil certification to DO-178B.
- Military certification to UK Def-Stan 00-55 involving an extensive, retrospective independent V&V (IV&V) activity.

For the main mission computer software, Lockheed adopted a well-documented correctness-by-construction approach [14, 15, 16]. The approach was based on:

- Semi-formal specifications using Consortium Requirements Engineering (CoRE) [17] and Parnas tables [18].
- "Thin-slice" prototyping of high-risk areas.
- Template-driven approach to the production of similar and repetitive code portions.
- Coding in SPARK with tool-supported static analysis carried out as part of the *coding* process and certainly prior to formal certification testing; this combination was sufficient to eliminate large numbers of errors at the coding stage – before any formal review or testing began.

This logical approach brought Lockheed significant dividends. Perhaps most striking was in the reduced cost of the formal testing required for DO-178B Level A certification: "Very few errors have been found in the software during even the most rigorous levels of FAA [Federal Aviation Administration] testing, which is being successfully conducted for less than a fifth of the normal cost in industry." At a later presentation [2] Lockheed was even more precise on the benefits claimed for their development approach:

- Code quality improved by a factor of 10 over industry norms for DO 178B Level A software.
- Productivity improved by a factor of four over previous comparable programs.
- Development costs were *half* that typical for non safety-critical code.
- With re-use and process maturity, there was a *further* productivity improvement of four on the C27J airlifter program.

These claims are impressive but they are justified by the results of the UK Ministry of Defense's own retrospective IV&V program that was carried out by Aerosystems International at Yeovil in the UK. It should be remembered that the code examined by Aerosystems had already been cleared to DO-178B Level A standards, which should indicate that it was suitable for safety-critical flight purposes. Key conclusions of this study follow:

- Significant, potentially safety-critical errors were found by static analysis in code developed to DO-178B Level A.
- Properties of the SPARK code (including proof of exception freedom) could readily be proved against Lockheed's semi-formal specification; this proof was shown to be cheaper than weaker forms of semantic analysis performed on non-SPARK code.
- SPARK code was found to have only 10 percent of the residual errors of full Ada; Ada was found to have only 10 percent of the residual errors of code written in C. This is an interesting counter to those who maintain that choice of programming language does not matter, and that critical code can be written correctly in any language: The claim may be true in principle but clearly is not commonly achieved in

practice.
- No statistically significant difference in residual error rate could be found between DO-178B Level A and Level B code, which raises interesting questions on the efficacy of the MC/DC test coverage criterion.

Lockheed succeeded because they had a strong process with an emphasis on requirements capture and accurate specifications; furthermore, they made their process repeatable by using templates. All these are wise things recommended by David Cook [3] in his article.

SPARK's role in this process was crucial. Its precision helped expose any lack of clarity in the specifications to be implemented. The exact semantics of SPARK require software writers to think carefully and express themselves clearly; any lack of precision is ruthlessly exposed by the rigorous analysis performed by its support tool, the SPARK Examiner[4].

One of the first effects of the adoption of SPARK on the project was to make the CoRE specification much more of a central and "live" document than before. Instead of "interpreting" unclear requirements on their own, developers were nagged by the Examiner into getting the required behavior agreed upon and the specification updated to match; everyone gained from this.

The second SPARK contribution came with the UK military certification process. Here the fact that SPARK facilitates formal verification or "proof" came into play. As Lockheed put it: "The technology for generating and discharging proof obligations, based on SPARK ... was crucial in binding the code to the initial requirements."

The Lockheed process might be termed "verification-driven development" since it is a process that recognizes that showing the system to be correct is usually harder than making it correct in the first place. Therefore the process is optimized to produce the evidence of correctness as a by-product of the method of development.

The SPARK Language

SPARK is an annotated subset of Ada with security obtained by two complementary approaches. First, the more problematic parts of Ada, e.g., unrestricted tasking, are eliminated. Second, remaining ambiguities are removed by providing additional information in the form of annotations. Annotations are special comments that make clear the programmer's intentions. Since they are comments, the compiler ignores them, but they are read and cross-checked against the code by the SPARK Examiner. Annotations also serve an important purpose by providing stronger descriptions of the abstractions used; this means that it is possible to analyze a part of the system without needing to access all the package bodies. Again this facilitates early analysis.

SPARK is introduced here to illustrate how a precise, unambiguous programming language is constructed and the benefits it brings. The goals set out for the original authors, more than 10 years ago, were as follows.

Logical Soundness

This covers the elimination of language ambiguities as mentioned earlier. The behavior of a SPARK program is wholly defined by and predictable from its source code.

Simplicity of Formal Language Definition

In order to demonstrate SPARK's logical soundness, it was felt desirable to write a formal definition of its static and dynamic semantics. This challenging task was completed in 1994, fortunately without producing any nasty surprises.

Expressive Power

The aim here was to produce a language that was rich and expressive enough for real industrial use. You can of course produce a safe subset of any language if you make it so small that it is impossible to write real programs in it: If you cannot write anything, you certainly cannot write dangerous code!

SPARK retains all the important Ada features required for writing well-engineered object-based code. It has packages, private types, functions returning structured types, and all of Ada's control structures. This leaves a clean, easy-to-learn language that, while smaller than Ada, is still rich and expressive.

Security

SPARK has no unenforceable language rules. The static semantic rules of SPARK are 100 percent machine-checkable using efficient analysis algorithms. It is this feature that makes it feasible to consider static analysis to be part of the design and coding process rather than seeing it as a retrospective V&V activity.

Verifiability

This is a consequence of the exact semantics of SPARK. Since the source code has a precise meaning, it is possible to reason about in a rigorous mathematical manner. The SPARK Examiner can be used to facilitate proof that SPARK code conforms to some suitable specifications, or that it has certain properties. A very straightforward and useful facility is the ability to construct a proof that the code will be completely free from run-time errors (such as the predefined exceptions).

Bounded Space and Time Requirements

A very important property of many critical systems is that they should operate for long periods; this means, for example, that they should not suffer from such things as memory leaks. SPARK programs are inherently bounded in space – there is no recursion or heap allocation for example – and can be made bounded in time. The end result is that the required machine resources can be calculated statically.

Correspondence With Ada

This is how the "Newspeak" trap is avoided. All SPARK programs are legal Ada programs and can be compiled with any standard compiler. More usefully, the meaning of a SPARK program cannot be affected by the implementation freedoms that the Ada standard allows the compiler writer. For example, it does not matter whether the compiler passes parameters by reference or by copying, or in which order it evaluates expression; the compiled SPARK code will have the same behavior. In effect, to use the terminology of the Ada LRM, it is not possible to write *erroneous* Ada programs in SPARK.

Verifiability of Compiled Code

Since we are taking advantage of the precision of the language to reason about source code, we need to consider the accuracy with which the compiler will generate object code. Clearly SPARK cannot change the behavior of compilers but there is some evidence that the simplifications of Ada provided by SPARK tend to exercise the well-trodden paths of a compiler rather than its obscure back alleys. The resulting machine code seems to be easier to relate to the source than might otherwise be the case.

Minimal Run-Time System Requirements

This is an extremely important area. Complex languages that provide facilities for concurrency, exception handling, etc., require large run-time library (RTL) support. Since the RTL forms part of the overall system, we need to demonstrate its correctness just as we must the application

code itself. As the RTL is likely to be proprietary and opaque, this can be very difficult.

SPARK inherently requires very little run-time support; for example, the SHOLIS [19] system developed by Praxis Critical Systems made only *one* call to a small routine in a fixed-point math library. Many Ada compiler vendors supply small RTLs for certification purposes, and SPARK is compatible with all of these. The smallest of all is GNAT Pro High-Integrity Edition from ACT because this has no RTL at all. As a demonstration, the SHOLIS code was ported to this system [20].

As an aside, it is quite interesting to compare the language subsets supported by the compiler vendors with SPARK. The former are all produced by removing the complex and difficult-to-understand parts of their run-time systems and seeing how much of the language can still be supported. Reasoning about the semantics of the language itself and eliminating problematic areas makes the latter. Both approaches produce almost identical subsets.

Cost Saving Example

So how did SPARK help Lockheed reduce its formal FAA test costs by 80 percent? The savings arose from avoiding testing repetition by eliminating most errors before testing even began. During the early stages of the project, before SPARK was fully adopted, an illustrative incident occurred. This is described in [15], which states: "In one case, a code error in a standard Ada module had resisted diagnosis for one week using normal testing methods. The module was converted to SPARK ... and the error was then identified in 20 minutes through SPARK analysis. Efficiencies of this type were obtained repeatedly."

Later in the project, when SPARK was fully established, the savings were even greater because errors of this kind never got to the test stage at all. They were eliminated as part of the coding process.

The case in question took the form of a Boolean function with a significant number of possible execution paths. On one of these paths there was a *data flow error* resulting in a random value being returned for the function. Finding this by test was extremely difficult because it was not enough simply to traverse the faulty branch. It was also necessary to be lucky with the random value returned. Since most random bytes of memory are non-zero, and non-zero values are typically regarded as being Boolean "true" by the compiler, the function usually returned true when the incorrect branch was executed; unfortunately, this was the right answer for this test condition! So the function nearly always behaved correctly but, apparently inexplicably, returned the wrong answer under seemingly random conditions. Simplified, the example was as follows:

- **Specification Table.** Table 1 represents the required behavior of an aircrew alert system. The mon column shows the input value of a "monitored" variable, in this case showing the severity of an alert message. The con column shows the required output of a "controlled" variable, in this case a Boolean value saying whether an alarm bell should sound in the cockpit.
- **Flawed Implementation.** A flawed implementation of the function might be:

```
type Alert is (Warning, Caution, Advisory);
function RingBell (Event : Alert)
return Boolean
is
      Result : Boolean;
begin
      if Event = Warning then
            Result := True;
      elsif Event = Advisory then
            Result := False;
      end if;
      return Result;
end RingBell
```

- **SPARK Examination.** The analysis performed by the SPARK Examiner includes a mathematically rigorous data and information flow analysis [21] that uncovers *all* uses of undefined data values thus:

```
 13 function RingBell (Event : Alert)
    return Boolean
 14 is
 15   Result : Boolean;
 16 begin
 17   if Event = Warning then
 18         Result := True;
 19   elsif Event = Advisory then
 20         Result := False;
 21   end if;
 22   return Result;
                ^1
??? ( 1) Warning : Expression contains
    reference(s) to variable Result,
    which may be undefined.
 23 end RingBell;
??? (2) Warning : The undefined initial
    value of Result may be used in the
    derivation of the function value.
```

This clear indication is obtained *at the engineer's terminal* before compilation, before test, before the code is even checked back into configuration management. The error never enters the system so it never has to be found and eliminated.

mon_Event	con_Bell
Warning	True
Caution	True
Advisory	False

Table 1: *Specification Table*

Conclusion

Most high-integrity and safety-critical developments make use of language subsets. Unfortunately, these subsets are usually informally designed and consist, in practice, of simply leaving out parts of the language thought to be likely to cause problems. Although this shortens the length of rope with which the programmers may hang themselves, it does not bring about any qualitative shift in what is possible.

The use of coherent subsets free from ambiguities and insecurities does bring such a shift. Crucially it allows analysis to be performed on source code before the expensive test phase is entered. This analysis is both more effective and cheaper than manual methods such as inspections. Inspections should still take place but can focus on more profitable things like "does this code meet its specification" rather than "is there a possible data-flow error."

Eliminating all these "noise" errors at the engineer's terminal greatly improves the efficiency of the test process because the testing can focus on showing that requirements have been met rather than becoming a "bug hunt." In my 10-plus years of using SPARK, I have never needed to use a debugger. I have become so used to things working the first time that my debugging skills have almost completely atrophied. The only price I pay for this is the SPARK Examiner pointing at the source code on my terminal and displaying messages telling me I have been stupid again; I find I am grateful for those messages!

The SPARK language definition [11, 12] is both free and freely available (see <www.sparkada.com> or e-mail sparkinfo@praxis-cs.co.uk). Alternatively, John Barnes' textbook [13] provides an informal and approachable description of the language together with demonstration tools.◆

References

1. Private communication arising from a productivity study at a major aerospace company.
2. Sutton, James. "Cost-Effective Approaches to Satisfy Safety-critical Regulatory Requirements." Workshop

Session, SIGAda 2000.
3. Cook, David. "Evolution of Programming Language and Why a Language is not Enough." CROSSTALK Dec. 1999: 7-12.
4. Littlewood, Bev, and Lorenzo Strigini, eds. "Validation of Ultrahigh Dependability for Software-Based Systems." CACM 1993, 36(11): 69-80.
5. Butler, Ricky W., and George B. Finelli, eds. "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software." IEEE Transactions on Software Engineering 19(1): 3-12.
6. Littlewood, B. "Limits to Evaluation of Software Dependability." In Software Reliability and Metrics. Proceedings of Seventh Annual CSR Conference, Garmisch-Partenkirchen.
7. RTCA-EUROCAE. Software Considerations in Airborne Systems and Equipment Certification. DO-178B/ED-12B. 1992.
8. Quoted in Andrew Hodges. Alan Turing: The Enigma. Walker & Co., ISBN 0-802-77580-2.
9. Motor Industry Research Association. Guidance for the Use of the C Language in Vehicle-Based Software. Apr. 1998, <www.misra.org.uk>.
10. Yu, Weider D. "A Software Fault Prevention Approach in Coding and Root Cause Analysis." Bell Labs Technical Journal Apr.-June 1998.
11. Finnie, Gavin, et al. "SPARK - The SPADE Ada Kernel." 3.3 ed. Praxis Critical Systems, 1997, <www.sparkada.com>.
12. Finnie, Gavin, et al. "SPARK 95 - The SPADE Ada 95 Kernel." Praxis Critical Systems, 1999, <www.sparkada. com>.
13. Barnes, John. High Integrity Ada - the SPARK Approach. Addison Wesley Longman, ISBN 0-201-17517-7.
14. Sutton, James, and Bernard Carré, eds. "Ada, the Cheapest Way to Build a Line of Business." 1994.
15. Sutton, James and Bernard Carré, eds. "Achieving High Integrity at Low Cost: A Constructive Approach." 1995.
16. Croxford, Martin, and James Sutton, eds. "Breaking through the V&V Bottleneck." Lecture Notes in Computer Science Volume 1031. Springer-Verlag 1996.
17. Software Productivity Consortium <www.software.org>.
18. Parnas, David L. "Inspection of Safety-Critical Software Using Program-Function Tables." IFIP Congress Vol. 3:270-277, 1994.
19. King, Hammond, Chapman, and Pryor, eds. "Is Proof More Cost-Effective than Testing?" IEEE Transaction on Software Engineering Vol. 26 No. 8:675-686.
20. Chapman and Dewar. "Reengineering a Safety-Critical Application Using SPARK 95 and GNORT." Lecture Notes in Computer Science Vol. 1622, Gonzales, Puente, eds. Springer-Verlag, Ada Europe 1999.
21. Bergeretti and Carré. "Information-Flow and Data-Flow Analysis of While-Programs." ACM Transactions on Programming Languages and Systems, 1985.

Notes

1. DO-178B (ED-12B in Europe) is the prevalent standard against which civil avionics software is certified. Level A software is defined as software whose failure would prevent continued safe flight or landing.
2. Static analyzers are tools that determine various properties of a program by inspecting its source code; unlike dynamic testing the program is not compiled and run.
3. The SPARK programming language is not sponsored by or affiliated with SPARC International Inc. and is not based on SPARC™ architecture.
4. SPARK is a programming language; the SPARK Examiner is the static analysis tool used to analyze SPARK programs.

About the Author

Peter Amey is a principal consultant with Praxis Critical Systems. An aeronautical engineer, he migrated to the software world when most of the interesting parts of aircraft started hiding in black boxes. Before joining Praxis, Amey served in the Royal Air Force and worked at DERA Boscombe Down on the certification of aircraft systems. He is a principal author of the SPARK Examiner tool and has wide experience applying SPARK to critical system development. Praxis Critical Systems will be exhibiting and presenting at the STC 2002.

Praxis Critical Systems
20, Manvers Street
Bath, BA1 1PX, UK
Phone: +44 (0) 1225 466991
Fax: +44 (0) 1225 469006
E-mail: peter.amey@praxis-cs.co.uk

Modeling and Simulation CMMI: A Conceptual View

Frank Richey
Illinois Institute of Technology Research Institute

There is a lack of guidelines for determining an organization's modeling and simulation (M&S) competency level such as maturity or capability. This makes it difficult for a customer to identify innovative companies who successfully deliver M&S-intensive products better, faster, and cheaper than the competition. This article advocates inclusion of M&S as a discipline into the Software Engineering Institute's Capability Maturity Model® Integration℠. This enhancement should provide the most efficient and cost effective means to identify and or improve a company's capability to manage the development, acquisition, and maintenance of M&S products and services.

Where can you find official acknowledgements or certifications that distinguish modeling and simulation (M&S) organizations as mature, capable, reputable, or accomplished? Development of both the modeling and simulation profession and the industry is inhibited by the fact that there is no generally accepted set of inherent qualifications or functional competencies. The lack of guidelines for determining M&S professional competency makes it difficult to establish and deliver educational programs. Furthermore, the lack of M&S metrics and standards for functional competency makes labor market transactions inefficient for both buyers and sellers of M&S professional services.

How can acquirers ensure their success by identifying contractor(s) who can provide M&S intensive systems that are delivered and maintained with predictable and improved cost, schedule, and quality? We are aware that M&S systems are software driven. However, can acquisition professionals be assured that any accomplished software development organization is capable of providing a quality M&S product within the projected cost and schedule? If not, then perhaps some consideration should be given to ways in which an M&S development organization can be recognized for its M&S capability.

The Software Engineering Institute's (SEI) Capability Maturity Model® Integration℠ (CMMI℠) can serve as a means for identifying a company's process maturity level in software, systems engineering, and acquisition. It could be of greater use to the members of the Department of Defense (DoD) M&S community and to the M&S community at large if M&S were included (see Figure 1). Also, it would be helpful to be able to recognize, among the high maturity level software development organizations, which ones are proficient as M&S developers and in which area of M&S their expertise resides.

Should Modeling and Simulation be an Integrated Component?

The Capability Maturity Model Integration provides:
- An ordered collection of best practices.
- A description of "WHAT" is required, not "WHO" or "HOW."
- A road map for the technical discipline and management process improvement.

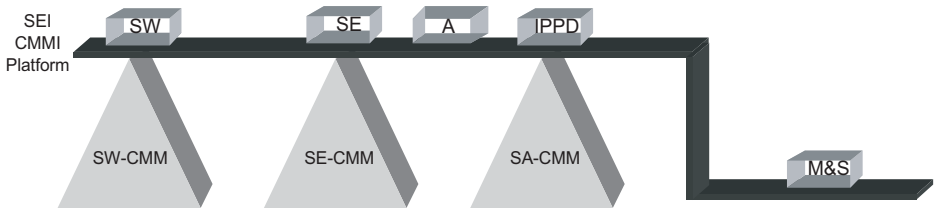


Figure 1: *Integrated Product and Process Development (IPPD)*

M&S Industry Standards and Best Practices

An industry standard provides an accepted technical approach for the design and development of communication systems, system architectures, and information systems. Industry standards for the life-cycle development of M&S systems do not yet exist. By conforming to an international standard for M&S design and development, organizations are confident that their M&S system modifications for enhancements, interface definitions, and interoperability can be realized more easily, cohesively, and cost effectively.

SEI has a large DoD and commercial audience, and its' CMMI model provides a product suite that offers an infrastructure that can be used to champion an industry standard for the design and development of M&S products and services. The designated standard should be cohesive and conform to its' associate disciplines, software, and system engineering (ISO/IEC 15504). The CMMI model's "Expected" and "Informative" component areas offer generic and specific practices, sub-practices, notes, and discipline-specific amplifications that are an existing structure that can be extremely beneficial for specifying industry standards and best practices for building M&S products.

Standards are important and beneficial because they refer to the formal mandatory requirements developed and used to prescribe consistent approaches for design and development of products (e.g., ISO/IEC standards, Institute of Electrical and Electronic Engineers standards, etc.). M&S supported by an international discipline standard would be equipped with the technical means to foster collaborative environments among different organizations or nations. These collaborative environments could address the interoperability of M&S systems between and among the military services, DoD government, non-DoD government, commercial, and international organizations.

SEI states that its "Product Suite represents a consensus-based approach to identifying and describing best practices in a variety of disciplines." We cannot think of a more convenient and appropriate vehicle to house and provide reference material for M&S industry best practices and standards.

M&S CMMI Activities

The Defense Modeling and Simulation Office (DMSO) has established an XMSO M&S CMMI Steering Group whose charter is to represent the M&S community in determining the feasibility of adding M&S as a discipline to the CMMI. It will also establish the

strategic direction and plan for the evolution and development of M&S process areas, practices, and amplifications for the CMMI Product Suite. The steering group will review and propose industry standards for M&S development, maintain the M&S CMMI requirements baseline (A-Specification) and Concept of Operations (ConOps), and ensure satisfaction of these requirements.

The plan is to extend the M&S CMMI Steering Group beyond the Defense Services Modeling and Simulation Offices (Defense Modeling and Simulation Office, Army Model and Simulation Office, Marine Corps Modeling and Simulation Management Office, Navy Modeling and Simulation Management Office, and Air Force Agency for Modeling and Simulation) to include the M&S DoD government, non-DoD government, and commercial communities. Any organizations with comments, ideas, or interest in participating in the steering group meetings may contact the author directly.◆

About the Author

Frank Richey is a senior software engineer for Illinois Institute of Technology (IIT) Research Institute, Alexandria, Va., providing a broad spectrum of experiences at the senior Systems and Software Engineering level. Currently, he is exploring the feasibility of defining an industry benchmark for assessing the credentials, capabilities, and organizational maturity of companies that provide modeling and simulation products and services for the Defense Modeling and Simulation Office. He is a member of the Institute of Electrical and Electronics Engineers and the Washington, D.C., area Software Process Improvement Network. He recently presented his white paper, "Modeling and Simulation Capability Maturity Model Exploration," at the fall 2001 Simulation Interoperability Workshop. Richey's experience includes the Internal Revenue Service modernization and 13 years of continuous Department of Defense experience. He has a bachelor's degree in computer and information science from Temple University.

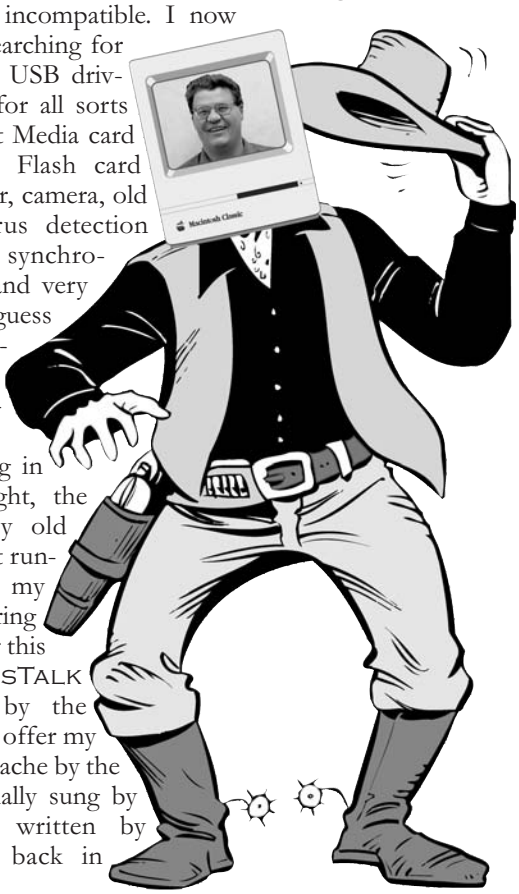
IIT Research Institute
1701 N. Beauregard Street
Alexandria, VA 22311
Phone: (703) 575-3296
E-mail: frichey@iitri.org

Software by the Numbers, Updates by the Score!

Late last summer, my lovely and charming wife gave me the OK to go out and buy a digital camera. After considerable searching, I bought the camera in October, and my wife and I really enjoy using it. The problem was that my old black-and-white laser printer (circa 1994) was now totally inadequate. So for Christmas, we bought ourselves a color printer. The printer was labeled "plug and play," and had a Universal Serial Bus (USB) connection. "How hard could it be to install?" I asked. Well, turns out it was a "plug-and-play-around-with-it" device.

The printer came with no manuals (they were on an included CD) and had a single installation sheet. Basically, the sheet said to install the drivers, and then plug in the printer. Early one afternoon after Christmas, I started the installation. Attempt No. 1 – system locked up when I plugged in the printer. Reboot, watch ScanDisk. Attempt No. 2 – system locked up before I plugged in the printer. Reboot, watch ScanDisk again, and load the manuals from the CD. Find obscure footnote saying, "For initial software installation, the printer cannot be plugged into a USB Hub device." Attempt No. 3 – Plug printer directly into USB outlet. Results? Same as attempts No. 1 and No. 2, but a much more spectacular lockup. Reboot, (again watch ScanDisk) and reload CD manuals. Find even more obscure footnote saying, "Software may not install correctly if virus detection software is enabled." Attempt No. 4 – disable virus detection software. SUCCESS!! Time spent – two hours.

You know, shortly before Christmas, I had updated my operating system (OS). The OS installed easily enough but many of my drivers were incompatible. I now know all about searching for and downloading USB drivers and updates for all sorts of devices (Smart Media card reader, Compact Flash card reader, CD burner, camera, old laser printer, virus detection software, PDA synchronization device, and very old scanner). I guess two hours installing a printer was actually pretty quick.



As I was lying in bed late last night, the words to a very old country song kept running through my mind. Remembering that the theme for this issue of CROSSTALK was "Software by the Numbers," I now offer my version of "Heartache by the Numbers," originally sung by Ray Price and written by Harlan Howard back in 1959.

® Heartache by the Numbers, copyright Tree Publishing Company (BMI).

Software by the Numbers

Verse 1

Problem number one was when I installed you.
I thought Win 3.1 would be easy (– so they all say).
And problem number two was moving up to 95,
With DOS 6.22 I should have stayed.

Chorus (Version 1.0)

Now I've got software by the numbers, updates by the score.
I've got so many peripherals on my USB,
I just can't plug in any more.
They all should load and coexist,
so their documentation does agree.
Yet every time I update one, three others crash on me.

Verse 2

Problems after number two came so quickly,
Installing 98, ME, 2000 and XP.
With hopeful heart I search for compliant drivers.
The misery of locating and
updating drivers will be the death of me.

Chorus (Version 1.1 with rhythm backward compatible with chorus Version 1.0)

Now I've got software by the numbers, drivers by the score.
For each security flaw that I patch, hackers find 10 more.
Well I've got software by the numbers,
updating drivers never stops.
'Cause just as soon as I think I'm done
– my gosh-darn PC locks.

Verse 3

Upgrading and patching software just taxes my brain.
Constant version changes and
new releases just seem to be the rule.
You'd think that all the vendors would end our misery.
Why don't USB drivers have a configuration management tool?

Chorus (Version 2.0 incompatible with any previous chorus; please uninstall older version of chorus before singing this version of chorus.)

Now I've got drivers by the numbers, updates by the score.
I can't think of a task that I love less, or a task that I hate more.
Suffering from documentation that doesn't exist,
and manuals that I lack.
I'm going to dump my PC box and return to my old Mac.

This has the potential to be a great country song – it mentions misery, love, hate, hardship, death, and suffering. If I could just work in a reference to my dogs (see my January 2002 BACKTALK column) and a pickup truck, I am sure it would top the charts! See you at Software Technology Conference 2002!

—David A. Cook, Software Technology Support Center
david.cook@hill.af.mil

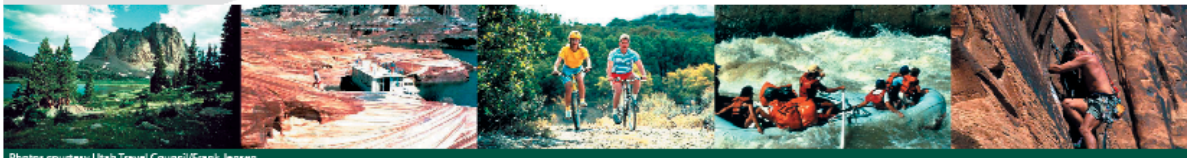
Dynamic Workshops Set in the Playground of the West

The Air Force's Software Technology Support Center (STSC) is sponsoring a series of workshops designed to help organizations build and buy software better. These informative workshops are **FREE** to U.S. government employees and will, for the second year, be held at the foot of the majestic Wasatch Mountains in state-of-the-art facilities in the vicinity of Hill Air Force Base.

The STSC workshop series continues throughout 2002 with these seven topics:

March 19-22	Process Improvement Using Theory of Constraints Thinking Process
April 24-26	Risk Management
May 21-23	Introduction to CMMI
June 18-20	Software Oriented Test and Evaluation
July 16-17	Software Best Practices: An Executive's Perspective
August 13-15	Object-Oriented Software Development: The Basics for Project Managers and Practitioners
September 17-19	Life-Cycle Software Project Management from Systems Engineering to Post Development Support

Be sure to catch this year's new workshops focusing on risk management, software oriented test and evaluation, best practices for executives, and the basics of object-oriented software development for project managers and practitioners. These workshops will fill up quickly and seating is limited. **Act quickly.**



Photos courtesy Utah Travel Council/Frank Jensen

For additional information visit our Web site at **www.stsc.hill.af.mil**.

SPACE IS LIMITED. To reserve your place at any of these workshops, contact Debra Ascuena at 801-775-5778 (DSN 775-5778) or debra.ascuena@hill.af.mil.

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

CrossTalk / TISE
7278 4th Street
Bldg. 100
Hill AFB, UT 84056-5205

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