

Experience Report

An organized, comprehensive metrics program can bring order to the chaos of small-project management and form the foundation for a concerted process improvement effort. The authors describe their experience in applying metrics to one such US Army organization.

Metrics for Small Projects: Experiences at the SED

Ross Grable, US Army Missile Command

Jacquelyn Jernigan, Casey Pogue, and Dale Divis, Tennessee Applied Physical Sciences, Inc.

The Software Engineering Directorate of the Research, Development, and Engineering Center at the US Army Missile Command designs, builds, and maintains small embedded applications. These projects consist of 10,000 to 50,000 lines of source code. Some SED projects also address larger systems in their sustainment phase, which often require modifications or enhancements to only a small percentage of the overall code per build.

Organizations such as the SED that produce and maintain many small embedded application software packages can benefit from a well-organized metrics program. Since the SED's emergence in 1984, software metrics have played an increasing role in the organization's development process. Before undertaking a concerted effort to coordinate its software metrics process, the SED had inaccurate, inconsistent data reported with no validation of usefulness or relevance. There was little respect or insight into the value of the metrics or measuring process, and virtually no return on the investment of time spent in data collection.

Although earlier attempts were unsuccessful, the SED has established an organization-wide metrics program that has been functioning successfully for over a year.

Most IEEE Software submissions are experience stories from academia or from fairly large companies. Articles from small companies are very rare, especially when it comes to experience reports on process improvement. This article is one of the few that describes attempts to introduce some process improvements in small companies.

Although the results are based on limited sample sets, they demonstrate that even in the smallest of development organizations a sensible approach to metrics can yield useful results. In this case, the results were helpful because the experiments were highly goal-oriented. The goals had a practical purpose and allowed the companies to learn something they could use for future improvements.

Provided metrics are collected with clear goals in mind, whether in large companies with sophisticated processes or in small ad hoc development environments, the results can give valuable insight into the success of improvement work. One lesson I learned from this article is that the mere existence of measurements leads to communication and discussion of results, which in turn enables teams to become more aware of key success factors in their work. Even if this was the only outcome, I believe it shows it is worth it to start taking some measurements.

—Wolfgang Strigel, From the Trenches editor

METRICS AT THE SED

In the SED's earlier days, development teams periodically performed panic collections of each day's code output for approximately 50 ongoing projects. The data collected was inconsistent, unreliable, and incomplete. After these early efforts, tactical missile projects began using standardized metrics for management briefings.¹ These metrics consisted of the number of people on the project, progress of the development phase, use of the target computer resources, analysis of schedule risk, resolution of trouble reports, number of delivered software projects, and supportability of software maintenance. Cooperation, completeness, and accuracy problems plagued the collection effort even though project leads presented some useful information in the quarterly progress review meetings. Scattered experiments performed on special pilot projects measured the complexity of, and effort devoted to, software developed in-house. However, the SED still lacked a coordinated effort to collect and analyze metrics data throughout the organization. Only with concerted effort and upper-management involvement did a useful software metrics program emerge.

These early attempts at metric collection, although they fostered no great revelations, taught the organization important lessons:

- ◆ Most project engineers routinely collected some kind of tracking data, yet resisted the imposition of centralized data collection.
- ◆ Project personnel expressed concern that the data collected would be used to assess individuals negatively.
- ◆ Project leaders expressed concern about the time taken from software development activities to collect metrics.
- ◆ Project leaders expressed concern about how the collected data would reflect poorly on

their projects when compared to the data from other projects.

- ◆ Management and policy makers appreciated and used simple presentations of analysis results, such as pie charts or red-amber-green traffic-signal graphics.
- ◆ Measurement of an activity, initially at least, caused improvement in the activity, as predicted by the Hawthorne Effect.
- ◆ Variations in metric definitions made it difficult to get consistent measurements.
- ◆ Software cost models produced good estimates for experienced engineers, even when there were no grounds for validation.

Software Engineering Evaluation System

In 1990, the SED developed a method of gathering metrics for validation and verification, called the Software Engineering Evaluation System. The SEES metric set consists of the number of people on the project, hours worked on the project, product items, planning, preparation, and defects. SEES records these metrics for software requirements, requirements traceability, software design and code, test plans, test descriptions, test witnessing, and functional- and physical-configuration audits. The SED provided training in SEES data collection and the reporting process for project leads involved with verification and validation of software, and developed a database for maintaining these metrics. The SED completed formal documentation of the SEES process in 1994.² Since then, a software group of the US National Aeronautics and Space Administration has also adopted this procedure for maintaining verification and validation metrics.

Metrics Working Group

The SED published an organization metrics policy in 1994 that established a Metrics Working Group

and laid the guidelines for collection and analysis of organization-wide metrics. This policy, still in use today, provides process audits and support funding. It requires that the information gathered be excluded from individual performance evaluations. This stipulation opened the door for integration of the metrics processes and procedures into the operational structure of the whole organization. The SED rewrote the standard operating procedures to include periodic collection and reporting of software metrics.

Overseeing the metrics initiative for the organization was the Metrics Working Group, which acts as a software metrics information center. Today the working group participates in professional metrics standards groups and leads research in metrics effectiveness. It maintains the metrics definitions for the organization, collects data from projects, and provides data analysis to management and project leads. Training provides SED personnel with an understanding of metrics policies and explains metrics definitions. The working group is also responsible for acquiring and maintaining the organization's metrics database.

The Metrics Working Group evaluated the SED's software-producing efficiency and calibrated a cost model for use in estimating future project costs.^{3,4} This task began with the bulk collection of a standard dataset for internal software development projects. The set included lines of code, engineering hours, calendar months, and the number of defects discovered in peer reviews and testing.

The collection process consisted of project leads completing a metrics form that contains the needed information. Scheduled interviews with each project leader answered questions about the data. The data collected demonstrated the diversity of the organization's projects: they varied significantly in size, effort, duration, and languages used. This data formed a baseline for the organization, one that helps indicate trends and areas of process improvement.

The analysis in the following sections lists projects anonymously and shows data for the organization as a whole, not by individual projects. The projects are listed anonymously to promote the honesty of reporting the data. Managers must get data on specific projects through the project leader, not

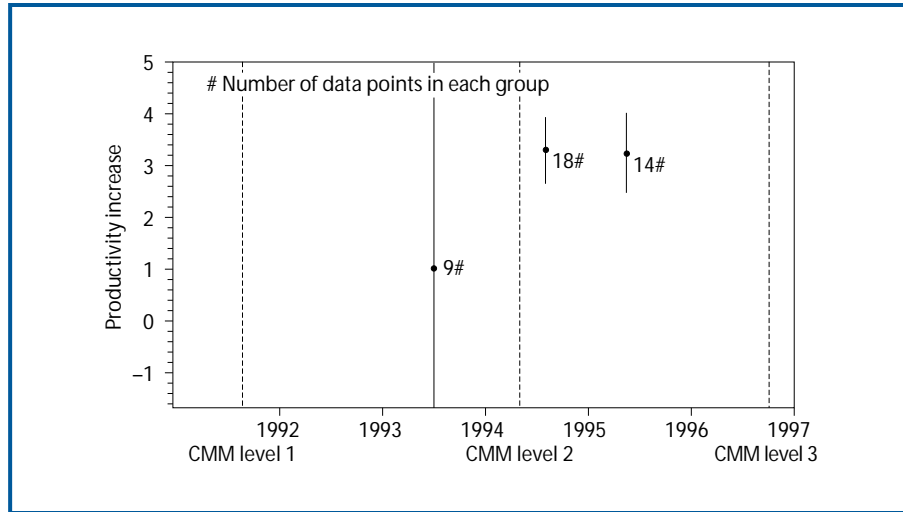


Figure 1. Annual productivity at the Software Engineering Directorate. The SED's productivity increased by a factor of approximately three between 1993 and 1994.

from the Metrics Working Group. Giving the project leads authority over their own data alleviated some of their concerns about project comparisons.

Along with these tasks, there have been various experimental data-collection projects. Typical data in these acquisitions include software science metrics, function points, feature points, state entropy, cyclomatic complexity, coupling and cohesion metrics, and cleanroom metrics.

ANALYSIS

The SED actively applies the SEI's Capability Maturity Model,⁵ which claims to increase the efficiency and productivity of an organization. The following analysis indicates that this claim is valid.

Efficiency analysis

The main goal of our efficiency analysis is to show productivity increases at the SED since the start of its software improvement efforts. Data collection began in January 1990, but data generated prior to 1993 is so sparse and unreliable that it was inadequate for establishing a baseline. Therefore, the efficiency analysis uses 1993 data to normalize each year's productivity mean. Figure 1 shows the productivity ratio, which is defined as each year's average productivity divided by 1993's average productivity. The data indicates that the SED's productivity increased by a factor of approximately three between 1993 and 1994. The productivity of SED projects completed in 1995 showed no significant change from the 1994 results. The large increase in productivity between 1993 and 1994 was primarily the result of appreciable software training during that interval. However, that very large productivity

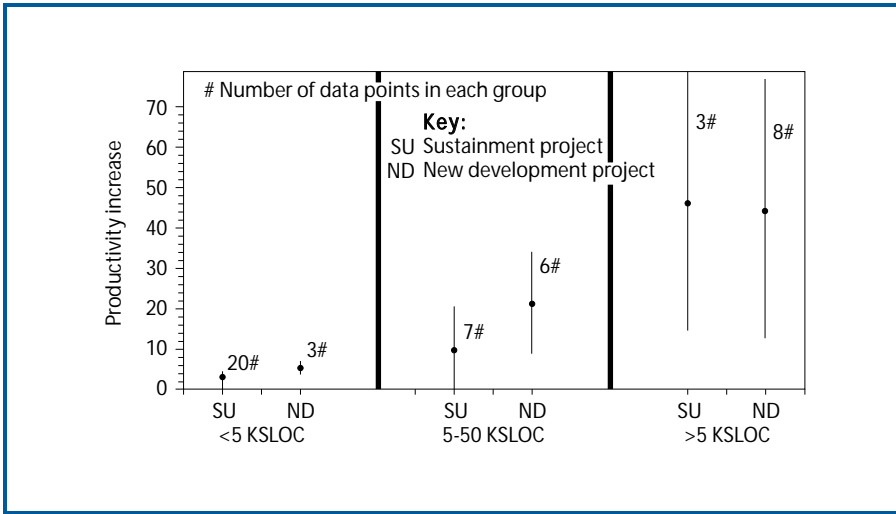


Figure 2. Average productivity for each project size category at the SED.

increase would not be expected year after year.

The productivity analysis partitioned the metrics data by size to investigate what effect project size would have on productivity. The analysis also divided the data between sustainment builds and new-development builds. Figure 2 shows the average productivity computed for each size category. The data indicates that the new-development builds are more efficient than the sustainment builds for the “very small” (less than 5,000 lines of code) and “small” (5,000 to 50,000 lines of code) size categories. The data also suggests that, for the SED, productivity increases for larger projects. This finding contradicts some experiments that show that productivity decreases for larger projects,⁶ and provides an example for why organizations should gather and analyze their own data. Not all organizations should follow the standard assumptions made by most software professionals. They should measure, analyze, and develop calibration models from their own software projects.

The statistical analysis included the use of small sampling theory since, in most datasets, the number of data points in a group numbered less than 30. The small sample numbers clearly contributed to the relatively large one-standard-deviation error shown in all the figures provided. In addition, the large differences in the characteristics of these small projects also contribute strongly to the large statistical deviations. In future work, we hope that sufficient numbers of projects in each year will allow the splitting of projects into subgroups that share similar characteristics. This may allow observation of statistical differences in productivity for different project categories.

Other factors that contributed significantly to errors in the calculations include the following.

- ◆ Project leaders obtained much of the data from

recordings made by people and not by unbiased machines. When people record data measurements on work they themselves performed, they often provide numbers in their favor or inaccurate values.

- ◆ Developers used several different languages to generate project code. Even though methods were used to normalize use of the various languages, large uncertainties persist.

- ◆ No one evaluated the education and experience levels of the contributors, even though there is general agreement that experience and education can exert a marked positive influence on productivity.

- ◆ Although project members recorded software error information, they did not use it to calculate the productivity values shown. This is important because a project may have relatively high lines of source code per effort value but also have a relatively high error rate. Thus, the effective productivity may not be high and certainly could be reduced.

Because of the diverse software being developed at the SED, the small number of builds with available data, and other limitations, determining conclusive results from this organizational analysis is difficult. However, we have identified several trends. Primarily, the analysis shows the importance of collecting accurate data and investigating trends in that data.

Cost model analysis

The cost model analysis investigated relationships between the metrics used in the cost models. This analysis computed linear and rank correlation coefficients and used nonlinear curve fits. The only metric pair with a “high” correlation coefficient was size (lines of code) and effort (0.80). However, this correlation does not appear in every project. Moreover, this relationship is not as strong in the sustainment builds (0.61) as in the new-development builds (0.84). This is because sustainment has much greater variability in requirements than new development.

A study of one project that measured individual code modules showed no correlation between effort and lines of code. This was, in large part, due to variations in module complexity. The project consisted of complex, tight hydraulic control loops and simple user interfaces. Nevertheless, a cost model based on estimated lines of code accurately

predicted project effort and schedule to within 10 percent. The number of actual lines of code was also within 10 percent of the estimated number.

Was the estimate a self-fulfilling prophecy? We think not, but will never know for sure. Verification of the reason behind the results requires more data than was available.

Barry Boehm's Constructive Cost Model⁷ (Cocomo) is one of the most popular software development cost-estimation models. However, the coefficients given in the Cocomo textbook are not valid for all organizations. Using the collected data, we calibrated the SED cost model analogous to the Cocomo basic model. The top graph in Figure 3 shows the effort versus the size of all SED projects. We measured effort in person-months, with each PM defined as 152 person-hours. The graph also shows the best nonlinear fit computed from the data, and the results obtained from the Cocomo basic organic model.

Table 1 lists the equations computed from several different data groups. The effort equation shows the relationship between the thousands of lines of codes (KSLOC) and the project person-months to generate the value of the KSLOC. The schedule or time-of-development equation (TDEV) shows the relationship between the total person-months and the total duration to complete the project. Significantly, the effort equations in the basic Cocomo model contain an exponent greater than one. When the exponential coefficient is greater than one, the model assumes that larger projects cost more per line of code than do smaller ones. Yet the effort equations from the SED data produced exponents less than one. The Cocomo model's "diseconomy of scale" did not ap-

Table 1
Software Equations Derived from SED Data

	Effort*	Schedule**
New-development projects	$PM^* = (2.22 \text{ KSLOC})^{0.55}$	$TDEV = (9.28 \text{ PM})^{0.154}$
Sustainment projects	$PM = (7.48 \text{ KSLOC})^{0.302}$	$TDEV = (2.28 \text{ PM})^{0.428}$
All projects	$PM = (5.39 \text{ KSLOC})^{0.368}$	$TDEV = (4.14 \text{ PM})^{0.316}$
"Very small" projects	$PM = (6.64 \text{ KSLOC})^{0.0723}$	$TDEV = (3.64 \text{ PM})^{0.169}$

*Computed as person-months needed to generate 1,000 lines of source code.

**Total development time, measured in person-months.

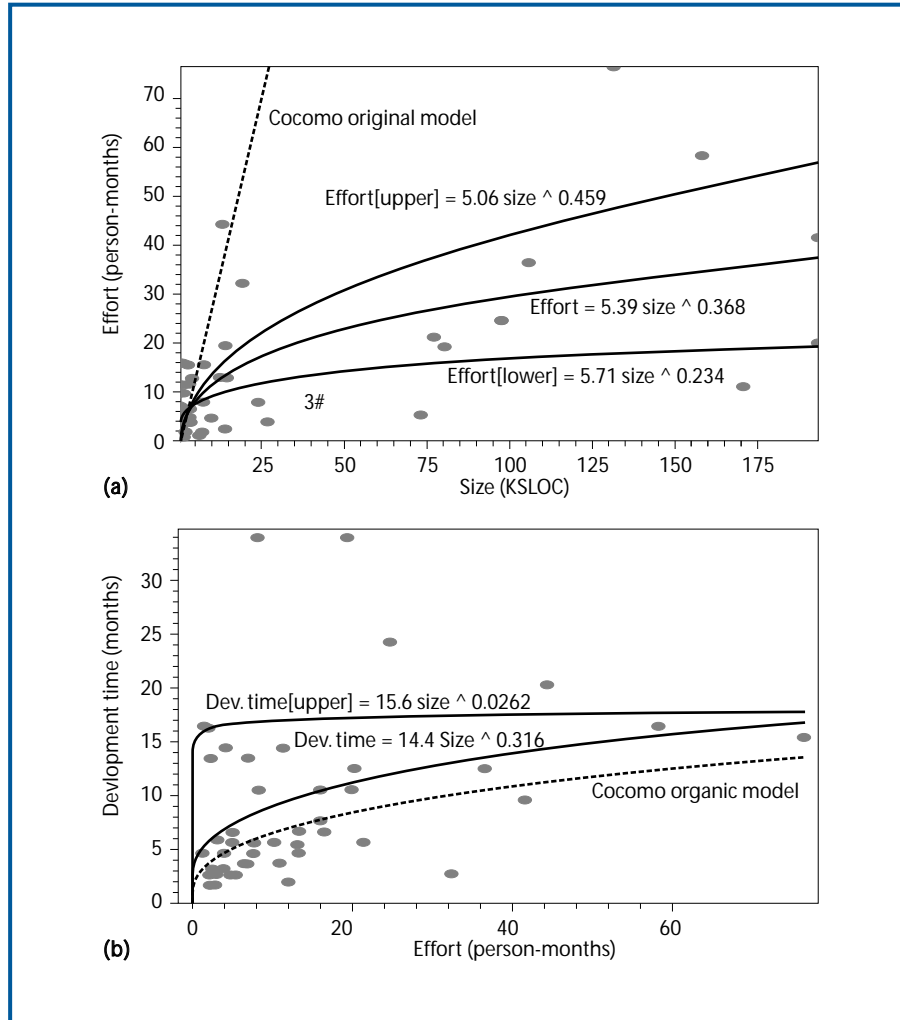


Figure 3. Comparing project data with the Cocomo organic model. The top graph (a) shows the effort in person-months vs. size for all SED projects; the bottom graph (b) shows the data related between development time and effort for all SED projects.

pear in our calibrated coefficients. With an exponent less than one (as for the SED projects), the model predicts that larger projects cost less per line of code than do small projects.

Figure 3 shows how data clusters near the origin, then spreads out as the project's size grows. Therefore, the analysis also calibrated the Cocomo

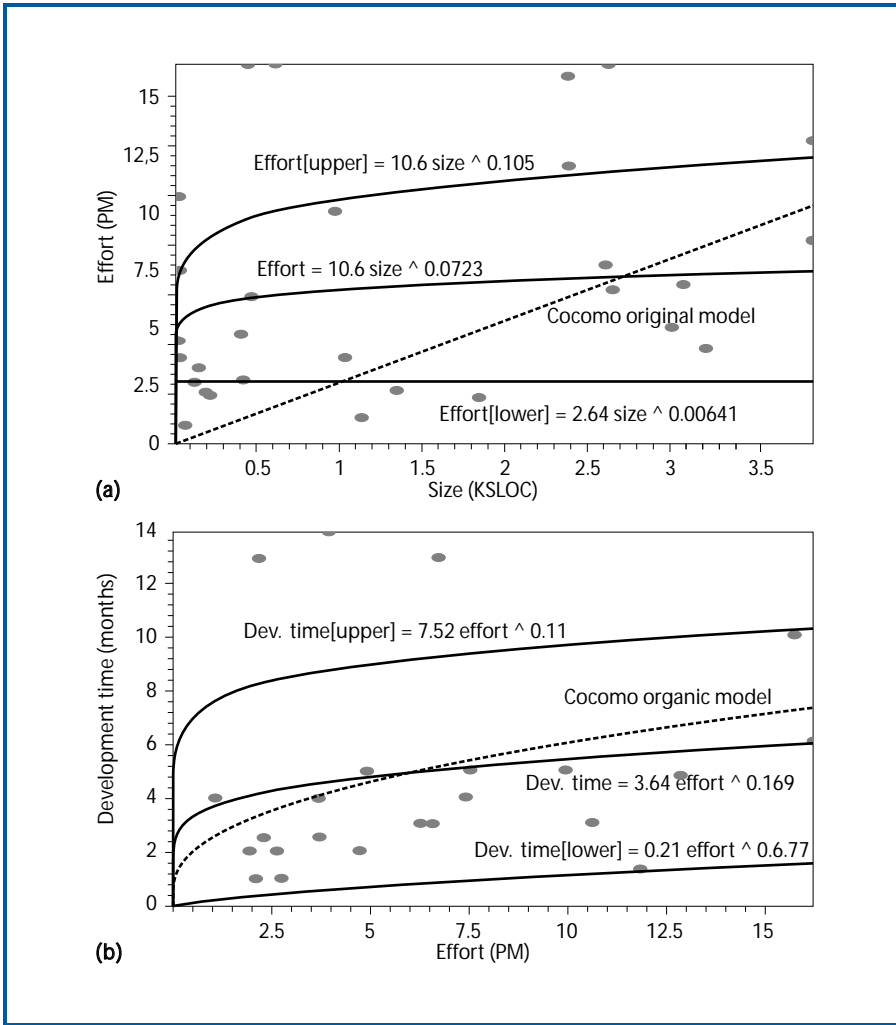


Figure 4. Comparing the very small project data with the Cocomo organic model. The top graph (a) shows the effort in person-months vs. size for all SED projects; the bottom graph (b) shows the correlation between development time and effort for all SED projects.

basic model for the “very small” projects (less than 5,000 lines of source code). Figure 4 shows the curves for the very small data.

The cost model analysis also computed upper- and lower-bound values for the calibrated cost model coefficients. In theory, these upper- and lower-bound coefficients should provide one standard deviation about the calibrated cost model. The upper- and lower-bound curves for the small projects contained approximately 53 percent of the SED builds. These curves also have exponents less than one.

The Cocomo basic schedule model estimated the SED builds much closer than did the effort model. The exponent for these curves was very close to the Cocomo model. The bottom graphs in Figures 3 and 4 show the SED duration in months as a function of effort hours. The upper and lower bounds of the small projects for the schedule data

contain approximately 87 percent of the SED builds.

The cost estimation analysis shows the importance of calibrating cost models. Using coefficients calibrated with data from other organizations can lead to large errors in software estimation. Textbook cost models give software practitioners a place to start; however, once data is available, analysts should compute new coefficients for their projects or organization. Metrics analysts should also update the cost model coefficients periodically to incorporate the most recent data collected.

CURRENT COLLECTION PROCEDURES

In February 1996, the SED implemented a standard procedure for the periodic collection and analysis of current-build metrics. The main goal of the SED Metrics Working Group in establishing this procedure was to make it minimally intrusive to project personnel while it gathered the needed data with more accuracy

and consistency than previous attempts. Before initiating the organization-wide collection procedures, the Metrics Working Group held detailed discussions on several topics, including

- ◆ metrics to collect,
- ◆ definitions of metrics,
- ◆ design of the metrics collection form,
- ◆ metric collection time period,
- ◆ person responsible for data collection, and
- ◆ analysis procedures.

The working group decided to begin with four basic metrics: effort, schedule, defects, and size, as recommended by the Software Engineering Institute.⁹ Substantial discussion and planning went into the design of the metrics collection form, incorporating suggestions from project personnel that would work for all SED projects. At first, the forms were confusing and complicated: they consisted of several sheets, with separate sheets for each metric

and a different sheet for each project classification. Based on suggestions from project leaders, the final collection form consisted of one page, with all four metrics and their definitions listed on the form's back.

The effort metric consists of hours directly charged to the project for 15 activities. Of these activities, 10 deal directly with the software development process, including requirement definition, design, code, and test. Four categories address project support activities, but are not directly involved with software development or sustainment; these include management and software quality assurance. An "other" category includes activities performed on the project that may not fit any of the previous categories.

The schedule metric consists of the planned date and actual date for 10 milestones, chosen to represent the main events in the software development process. A project's staff can tailor its process to have fewer milestones, in accordance with the SED procedures.

The defect metric represents the number of new and closed defects. The metric divides defects into three classes depending on the severity of the defect. Several projects use peer inspection reports for recording defects. Project leads can attach the peer inspection reports to the metrics collection sheet, which helps reduce their paperwork.

The size metric consists of the language, number of lines of source code developed or modified, and number of document pages generated. The sustainment builds also record the total lines of code in the system.

We decided to collect metrics biweekly for two reasons: government employees report their hours biweekly, and it is less intrusive than weekly collection but frequent enough to ensure the data's consistency and correctness. Project leaders collect and submit the forms to the working group, ensuring collection of all project data and helping to guarantee the data's accuracy.

The SED consists of approximately 300 government and contract personnel. Approximately 130 of

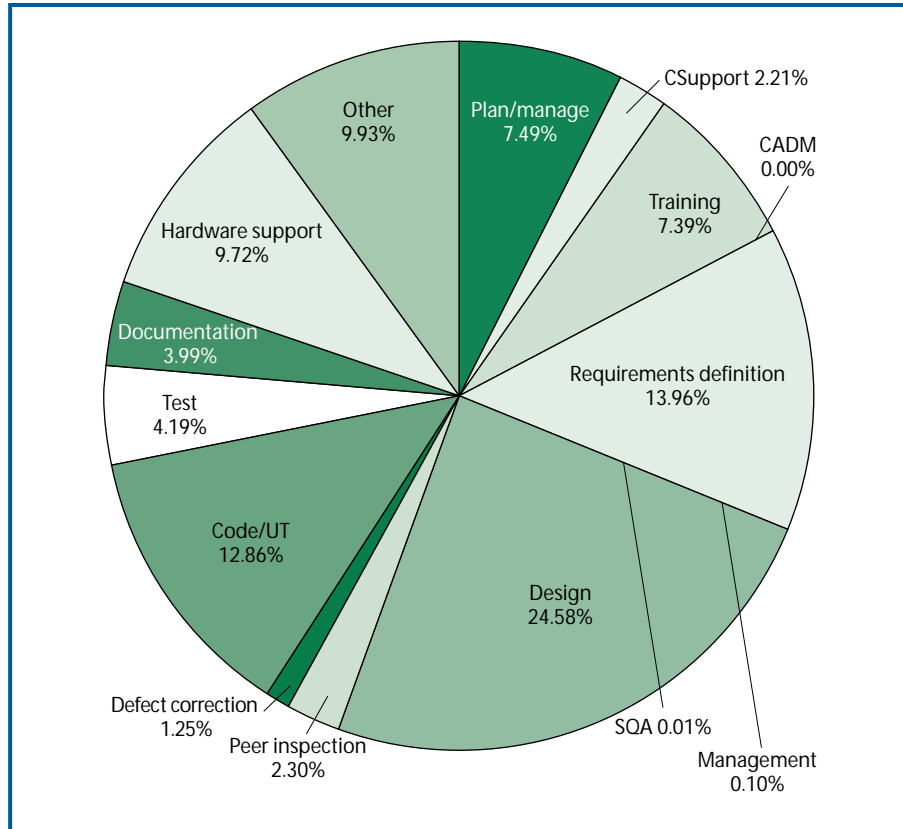


Figure 5. Sample monthly metrics report for an SED project.

these are software personnel who work on the 22 projects currently participating in the metrics collection program. Thus, within a two-week period, project leaders typically record 5,700 personnel hours on the metrics collection sheets. The number of hours collected varies, since project personnel are always changing and new projects begin as others end.

The Metrics Working Group generates monthly metrics reports for each project and sends them to the project leads. The reports consist of comparisons between actual and estimated values, percentages of effort hours spent in each category, and the number of defects open. Figure 5 contains a sample pie chart that shows the percentages of effort hours expended in each category. The pie chart summarizes the distribution of effort by activity in concise and easy-to-read form.

Other simple charts of size, defects, and schedule metrics also aid project leaders in managing their projects. These simple charts may also alert the leaders to potential problems early in the software development process and provide time for corrective action. The monthly metrics reports are still evolving; they change as project leads realize the management benefits this data provides.

Taking advantage of SED-funded training for several of its members, the Metrics Working Group began using a sophisticated database tool to

manage the data. However, even though the database had some powerful features, it proved difficult to learn and use. The working group discovered that simple, inexpensive database tools and spreadsheets could adequately produce the needed results.

FUTURE PLANS

The working group is developing a network-based collection system, and plans to phase it in in small steps. Several different implementation methods are being investigated.

The group also periodically updates the cost model analysis, and it is working on a spreadsheet that incorporates the calibrated coefficients into an estimation model for project leader use. This spreadsheet will incorporate the percentage data for each effort category from completed projects to help project leads estimate hours in each effort category. Calibrating the cost models for the organization, providing training, and providing metrics tools will help increase the accuracy of software project estimation.

The Metrics Working Group continues to research new metrics and models, investigating different ways of measuring complexity. Several other areas are also being investigated as the SED begins plans for reaching CMM level 4.

Upper management's involvement and the publication of an organizational metrics policy contributed to the success of the present metrics collection program at the SED. The organization learned many lessons in the process of establishing measured feedback for its processes, including the following:

- ◆ A successful metrics program must have support from the highest management.
- ◆ Training is essential for the correct collection and analysis of data, and for management to be able to appropriately apply metrics. All managers and engineers need training in the collection and use of the data.
- ◆ Although metrics use does lead to better organizational performance, it requires an up-front investment of 3 to 8 percent of the project's overall cost, which must be paid at the beginning of the metrics program. This cost drops to less than 1 percent after project personnel become familiar with the program.
- ◆ Experiments to validate and analyze the metrics are an essential process component. Analysis of the data can validate assumptions or show that typical software process assumptions do not apply to a particular organization or project.

- ◆ Periodic calibration of cost models helps ensure accurate cost estimation.

- ◆ The metrics analysis techniques used should be statistically sound to ensure the accuracy of conclusions drawn from data analysis.

- ◆ A metrics collection program should be initiated with a small set of metrics that will show the benefits of collecting data. The metrics collection form should have a simple and easily understood design; a complicated form will only discourage software personnel from collecting data.

- ◆ Establish a metrics feedback procedure for engineers and managers to learn how to use the metrics and to help them understand the use and benefits of the measurement system.

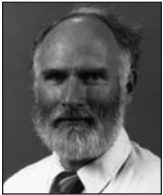
- ◆ Inexpensive commercial database and spreadsheet tools are all that an organization needs to track metric data and provide meaningful reports.

The SED continues to learn important lessons from the metrics process. Our organization monitors its process improvement program's progress using the baseline established from collected data. The metrics program continues to expand as SED personnel experience the benefits of collecting and analyzing software metric data. ❖

REFERENCES

1. "Methodology for the Management of Software Acquisition," Software Engineering Directorate, Research, Development, and Engineering Center of the US Army Missile Command, Redstone Arsenal, Ala., 1991.
2. "Software Engineering Evaluation System (SEES), Volume I, SEES Executive Summary," Software Engineering Directorate, Research, Development, and Engineering Center of the US Army Missile Command, SED-SES-IES-001, Redstone Arsenal, Ala., 1994.
3. J.G. Jernigan and D.H. Divis, "Metric Analysis of Historical Data for the Software Engineering Directorate," Tennessee Applied Physical Sciences, TAPS-22-1995, Sept. 1995.
4. J.G. Jernigan, C.R. Pogue, and D.H. Divis, "Software Metric Analysis for the Software Engineering Directorate 1996," Tennessee Applied Physical Sciences, TAPS-24-1996, Sept. 1996.
5. M.C. Paulk et al., "Capability Maturity Model for Software, Version 1.1," Software Eng. Inst., Carnegie Mellon Univ., CMU/SEI-93-TR-24 or ESC-TR-93-177, Pittsburgh, 1993.
6. L.H. Putnam and W. Myers, *Measures for Excellence: Reliable Software on Time, Within Budget*, Yourdon Press, Englewood Cliffs, N.J., 1992.
7. B.W. Boehm, *Software Engineering Economics*, Prentice-Hall, Upper Saddle River, N.J., 1981.
8. A.D. Carleton et al., "Software Measurement for DoD Systems: Recommendations for Initial Core Measures," Software Eng. Inst., Carnegie Mellon Univ., CMU/SEI-92-TR-19 or ESC-TR-92-019, Pittsburgh, 1992.

About the Authors



D. Ross Grable is on the computer science faculty at the Clinch Valley College of the University of Virginia. His research interests are software and system metrics, software project management, and information system measurement techniques. He has worked with the Army Aviation and Missile Command as a senior scientist

and consultant, and has also worked as a programmer, analyst, and project manager on many aerospace and military software-intensive real-time systems projects.

Ross received a BS in mathematics and physics, and an MS and a PhD in computer science from the University of Alabama Huntsville. He is a member of IEEE and the IEEE Computer Society.



Casey R. Pogue is a computer scientist at Tennessee Applied Physical Sciences. He has developed accounting tools for small businesses and worked in computer services support. His current work includes metric analysis, Ada Compiler evaluation, and code analysis for the Software Engineering Directorate

of the US Army Missile Command. His current research interests include software metrics and compiler optimization.

Pogue received a BS in computer science from Middle Tennessee State University.



Jacquelyn G. Jernigan is a weapon system analyst at Lockheed Martin Missile and Space in Huntsville, Alabama. Previously she was a senior software engineer at Tennessee Applied Physical Sciences. She has performed systems and radar analysis and developed analysis tools and radar simulations for government agencies for seven years. Her recent work includes metric analysis for the Software Engineering Directorate of the US Army Missile Command. Her current research interests include software metrics, process improvement, and system performance.

Jernigan received a BS in electrical engineering from the University of Alabama, Huntsville. She is a member of the IEEE Computer Society.



Dale H. Divis is president of Tennessee Applied Physical Sciences. He has performed physics and mathematical modeling research for government agencies, and has evaluated data from physical experiments with unique statistical methods. Currently, he is involved with the statistical evaluation of software metrics data.

Divis received a BA in physics with minors in mathematics and chemistry from Rutgers and a MS in physics from Texas Christian University. He is a member of the American Physical Society.