METRICS FOR COMPETITIVENESS

Rolland H. Berry
George H. Wedberg

Planning Analysis Corporation
8200 Greensboro Drive, Suite 810
McLean, Va. 22102

I. Introduction

After briefly discussing how metrics support the development of competitive software, this paper describes the use of specific software metrics currently applied to the development of the Standard Installation/Division Personnel System-3 (SIDPERS-3) for the US Army. The development is being done in Ada using a tailored Department of Defense System Software Development Military Standard (DoD MIL-STD 2167A) approach. The paper explains how the metrics were selected by the contractor team, and how they are being implemented.

II. Competitiveness and Quality

As the amount of funding for Department of Defense software development decreases, contractors will increasingly have to demonstrate competitiveness. The Deputy Secretary of Defense, Donald J. Atwood stated the situation clearly in May 1989:

> "The Department of Defense must continuously seek measures to increase productivity in the defense acquisition process to live within budget constraints without jeopardizing national defense and readiness... Those contractors who provide "Best Value" to the government by consistently demonstrating, through performance on production contracts, an ability to deliver on time while consistently improving quality and reducing cost should be rewarded for their accomplishments."[1]

Mr. Atwood extended the carrot in the quote above, but he also held up the stick:

> "DoD will continue to use competition... Well crafted competitions should not be looked upon as a threat, but rather, as opportunities for the most efficient and highest quality producers to gain and maintain increased DoD business. As we structure our competitions, past performance, including quality, cost and delivery should be more significant determinants in contract award decisions."[2]

Mr. Frank Carlucci, former Secretary of Defense, agrees and so states in a DoD policy memorandum: "Quality must be a key element of competition."[3] Clearly, competition is here to stay, and the way to be competitive is to produce quality.

What then is quality? No answer to that question could exclude the work of W. Edwards Deming, whose teachings about quality process and its relationship to competitiveness are well known. They are summarized in a Department of the Navy publication on total quality management (TQM) as follows:

- Quality is defined by customer requirements.
- Top management has direct responsibility for quality improvement.
- Increased quality comes from systematic analysis and improvement of work processes.
- Quality improvement is a continuous effort and conducted throughout the organization.[4]

TQM is the method by which an organization improves quality through continuous process improvement.[5] It is recognized that top management is key to quality improvement. However, the concern here is the quality improvement that comes from systematic analysis and improvement of work processes. Lawrence H. Putnam, identifies in IEEE Software, March 1991 the importance of metrics in TQM:

> "Throughout government and industry, the idea of total quality management has caught on as we face increasing competition in all segments

of the economy. The results of Japanese refinement of this philosophy for the last 40 years are clearly evident. So the concept is not new. What is new is applying these principles and those of W. Edwards Deming to the software management process... This means:

- taking quality seriously,
- ...
- measuring progress with the right metrics..."[6]

In software development, an important part of systematic analysis is a well designed and implemented software metrics program.

III. Metrics Program

This section outlines the basic assumptions, underlying the metrics program.

A properly designed metrics program measures a few critical aspects of the software development process in a non-invasive manner. The required metrics data should be obtainable without imposing a significant burden on the technical staff. Wherever possible, the metrics data should be obtained directly from the development process, rather than requiring a separate effort. The metrics program will be slighted by the technical staff if it is perceived to take too much time and effort away from the "real work" of the project.

For much the same reason, the number of metrics employed should be constrained. Although it is possible to measure dozens, if not hundreds, of project-related quantities, the trick is to select a small number of metrics that supply useful information and insights.

Metrics information is best obtained from data derived from technical reviews that are held as part of the normal development process, and from information derived from software testing results. Our implementation of this is that the data for the metrics used throughout the software requirements analysis, and design, code, and unit test phases are collected at technical reviews. Test data is collected during quality assurance testing.

A properly designed metrics program provides simple, direct indications to management about the quality of the process and the product, and the progress of the project. Direct information about significant, technical aspects of the software development process facilitates communication between the technical staff and the management of the project.

IV. Metrics Selection

For the SIDPERS-3 development effort, the project team was directed to use the metrics in Army Materiel Command Pamphlets 70-13 (Management Insight), and 70-14 (Management Quality Insight) or to propose an alternative metrics program. After examining the metrics in the pamphlets and consulting the literature, the Goal-Question-Metric[7] paradigm was identified as a useful method for defining metrics.

This method breaks up the metrics selection process into three steps. The first step is to list the goals of the metrics program. The second step is to formulate questions regarding the goals that will help to define relevant metrics. The third step is to identify relevant metrics. A one-to-one relationship does not necessarily exist between a goal, a question that characterizes it, and a single metric that answers the question and meets the goal. Some metrics may answer more than one question or pertain to more than one goal.

In the course of using the method, several feasible program goals, many possible questions, and a plethora of metrics may be produced. Some of the feasible goals that were considered are: an orderly software development process, visibility into cost schedule deviations, a stable staff, thorough understanding of the requirements, requirements traceability, unchanging requirements, good design, correct code, and error detection as early in the process as possible.

Four goals were selected.

1. Ensure clear understanding of the customer's requirements.
2. Track project progress.
3. Provide a basis for process and product improvements.
4. Produce a high quality product.

Having devised a manageable list of program goals, we asked the questions that suggested metrics to achieve the goals. Among the many possible questions produced as part of the analysis were: Is the software development plan being used effectively? Are we meeting our internal deadlines? What has been planned/promised? How correct is the code? How many defects are there in the code? How well do we understand the requirements? How stable are the requirements? When are the errors, faults, etc. detected? How many errors are caught in each inspection? Are we detecting errors as early as possible? When should we stop testing?

Analysis of the questions suggested eight key metrics for implementation.

1. Requirements definition.
2. Requirements stability.
3. Software progress.

4. Testing progress.
5. Product issues.
6. Product defects.
7. Test failures.
8. Test sufficiency.

V. Project Metrics

The remainder of this paper discusses the eight metrics listed above. Each metric is discussed as follows:

a. Use of the metric A brief synopsis of the use of the metric by the project team is presented.

b. Actual data or data format A discussion of the actual data gathered to date is presented and the charts used to portray the data are shown. If not enough data is available at the time of publication to illustrate the use of a metric, a sample chart is shown to illustrate the format in which the data will be presented.

1. Requirements Definition

a. Use of the metric This metric is used during the software requirements analysis phase. Its purpose is to get a relatively early indication that the requirements are well-understood by the project team. SIDPERS-3 customer requirements are in the form of narrative summaries, structured English, and data flow diagrams. The developers' task is to convert the customer requirements into a set of clearly defined functional capabilities required of the software product.

We consider a capability to be defined when it has been determined to be traceable and testable. A capability is considered traceable if it is possible for reviewers to trace that capability to a customer requirement. A capability is considered testable when it is sufficiently well understood to write a test description. The determination of requirements definition is made at technical reviews during the software requirements analysis phase of the project.

b. Actual data As shown in Figure 1, the chart to present the requirements definition graphically is a plot of three pieces of data versus time. The three are: estimated number of total capabilities (unshaded vertical box), number of traceable capabilities, and number of capabilities defined (both traceable and testable). Investigation of the varying height of the unshaded bar beginning in October revealed some uncertainty in the definition of functional capabilities, coupled with decisions to combine capabilities. Overall the chart shows acceptable progress in preparation for the informal requirements review planned for April.

2. Requirements Stability

a. Use of the metric The purpose of this metric is to show the number of changes to project requirements and to identify which capabilities change most often. The number of changes to requirements indicates the degree of requirements stability. The metric is employed only after the software requirements analysis phase is completed. The software requirements analysis phase ends with a requirements technical review. After the

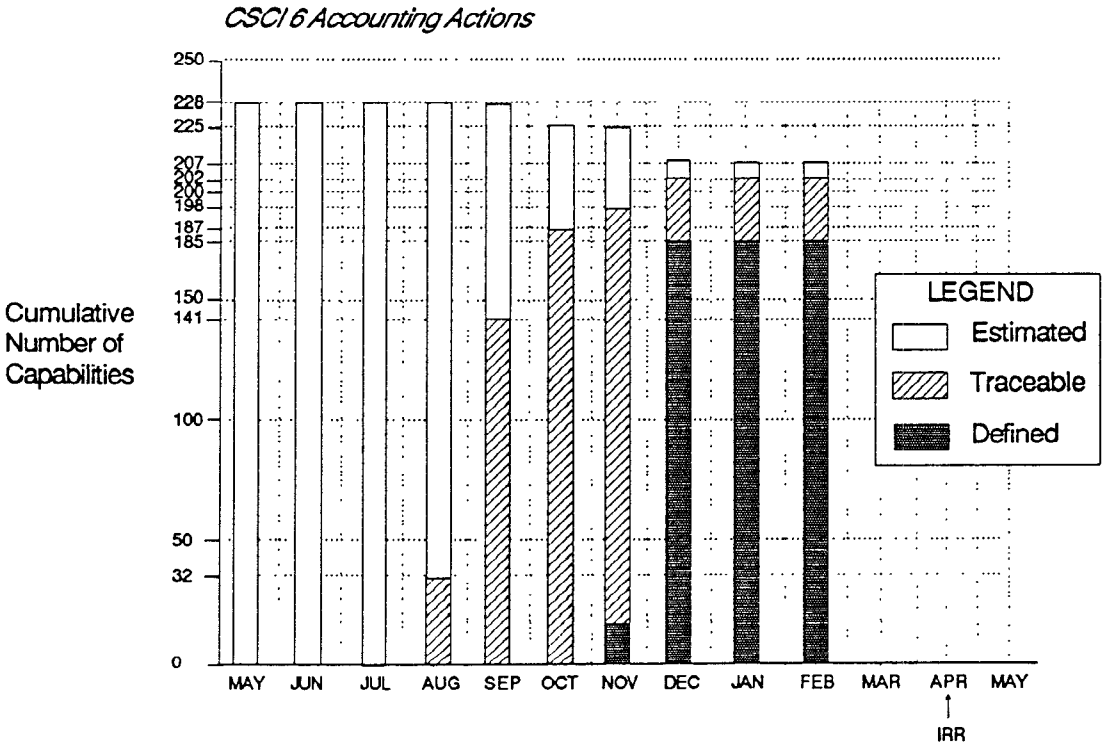

Figure 1. Requirements Definition Metric

path analysis has been performed to determine the longest thread through the path. The time constraints are useful as design goals and testing criteria.

```
--HARD Path: Monitor the PH of the solution
--Executes [PH_Monitor] on [OEM Computer]
--ACTIVATES PERIODICALLY AT 8 HZ
--C_critical = 100.0 US
--C_min = 10.0 US
--T = 125.0 MS
--U["OEM Computer"] =
--   MAX 0.080 PERCENT
--   MIN 0.008 PERCENT
--   AVE 0.044 PERCENT

ENTER WHEN((ON 8 Hz period))
--   BEGIN PATH DESCRIPTION
     --Scale PH sensor data to [1 .. 14]
     CONVERT_TO_STD_PH_SCALE( in PH_Sensor, out
     PH_VALUE);--Uses  (OEM Computer) 50.0 US
--CRITICAL PATH TIME: 50.0 US
     if((PH_VALUE<5.0))
       THEN
       --Handle PH alerts
       SET_ALERT( out PH_Alert);--Uses  (OEM
Computer) 50.0 US
     ELSE
       if((PH_VALUE>5.8))
         THEN
         --Handle PH alerts
         SET_ALERT( out PH_Alert);--Uses  (OEM
Computer) 50.0 US
       ELSE
        if((PH_VALUE>=5.0)
        AND (PH_VALUE<=5.8))
          THEN
          --Handle PH alerts
          RESET_ALERT( out PH_Alert);--Uses
          (OEM Computer) 50.0 US
          END_IF;
       END_IF
     END_IF;
--   END PATH DESCRIPTION
```

The statements in this PDL are the behaviors in the lowest level SADT activity boxes. Hard real-time behaviors must be implemented with ceilings on CPU use that do not exceed the specified usage in the PDL. Activation rules are simplified by CPATH in a manner similar to what a compiler's optimizer might do to simplify expressions. Activity names show up as comments. IF-THEN-ELSE and CASE structures are identified by the CPATH analyzer.

The present CPATH analyzer is still experimental. A good deal of development work is still going on to improve its control path analysis using optimization techniques. The present version separates soft and hard paths on individual processors and inserts operating system calls to manage the transition when a hard path calls a soft one, or vise-versa.

Slated for the future is the generation of tasking models which incorporate the threads. We plan on implementing heuristic approaches to both cyclic exec style process control and event driven control which take into consideration task coupling through shared data.

Experience.

SALDT has been operational in various forms for approximately 3 years. The Rate Monotonic Analysis capability was introduced in 1990. It has been applied to requirements, design and reverse engineering problems in both real-time and non-real-time situations. Preliminary trials with Ada PDL generation (with time constraints annotated in the PDL) have been successful in that coders have been able to carry out the fabrication from the SALDT specifications. Trials with the PDL have so far been limited to small applications (a few hundred lines of code, and in one case a system of seven programs.) One of the trials was contractual work requiring the development of example avionics code running at eight Hz on a MIL-STD-1750A machine (the language happened to be Jovial). Approximately 450 lines of code were written, including the process control code for managing an eight Hz cyclic process. The example was coded and tested on a simulator in 24 man hours starting from the PDL generated from an SALDT model of the software.

Our experience with the development of large Ada systems suggests that the value of SALDT in large real-time projects is probably not the benefits derived from PDL generation, although the ability to produce good quality PDL will indeed be a productive feature. We expect that the highest value is simply the ability to specify timing constraints appropriate for real-time engineering practice, and the corresponding ability of management to measure the completeness of this kind of a design through application of verification tools.

The tool set and methodology was introduced to Academia in the Autumn of 1990 in an undergraduate course in Software Engineering at the SUNY Institute Of Technology at Utica Rome. Students received 8 hours of lecture on Rate Monotonic Scheduling, 4 hours on the essentials of SADT, and four hours on SALDT methods and tools. They were then assigned a problem involving the computation of present position, velocity and heading from radio navigation data. Students were required to carry through a functional requirements model and a design model with real-time constraints. As is inevitable in an academic setting, some resourceful students recognized the potential to base lines of code estimates on SALDT models and performed their COCOMO cost projections on these estimates. Only about a third of the students, however, were able to make appropriate use of the SALDT methods and tools. Student feedback was consistent about needing more time and practice to acquire familiarity with the methods and tools.

The SALDT tools have been made available commercially and feedback has been received from users in the USA and in Europe. Real-time users have universally expressed the

need for an improvement in the editing capabilities for activation and behavior rules. (The present editing capability for these is "delete and try again.") There has been some acknowledged appreciation for the hierarchical consistency checking of activation rules. There has been some spirited appreciation for the automatic drawing feature and the validators.

References.

(1) Ross, D. T. "Structured Analysis (SA): A Language For Communicating Ideas," IEEE Transactions on Software Engineering, vol. 3, no. 1, January 1977, pp 16-34.

(2) L Liu, Layland, "Scheduling Algorithms For Multiprocessing In A Hard Real-Time System." J. ACM, 20(1), 1973.

(3) Lui Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols, An Approach to Real-Time Synchronization," Technical Report CMU-CS-87-181, Carnegie Mellon University (Nov 1987).

(4) Baker, T.P., "Preemption vs. Priority, and the Importance of Earily Blocking." "Real-Time Systems Newsletter, Volume 6, Number 2, IEEE Computer Society.

(5) Lui Sha, John B. Goodenough, "The Priority Ceiling Protocol: A Method for Minimizing The Blocking Of High Order Ada Tasks." Ada Letters, Special Issue: Proceedings of the 2nd International Workshop on Real-Time Ada Issues VIII, vol. 7, Fall 1988, pp 20-31.

(6) P. Puschner, R. Zainkinger, "Developing Software with Predictable Timing Behavior." "Real-Time Systems Newsletter, Volume 6, Number 2, IEEE Computer Society.

(7) L.D. Molesky, K. Ramamritham, C. Shen, J. A. Stankovic, and G. Zlokapa, "Implementing a Predicable Real-Time Multiprocessor Kernal - The Spring Kernel,"Real-Time Systems Newsletter, Volume 6, Number 2, IEEE Computer Society.

(8) D. A. Marca, C. L. McGowan, "Structured Analysis and Design Technique." McGraw Hill Book Company, New York, St. Louis, 1987. ISBN 0-07-040235-3.

(9) Wallace, W.H., Stockenburg, J.E., and Charette, R.N., "A Unified Methodology for Developing Systems." Intertext McGraw Hill. 1987. ISBN 0-070-010646-0.

(10) Software Engineering Institute, "Rate Monotonic Scheduling Theory Tutorial: Exercises and Case Studies. Tri Ada "90", 1990.

(11) J. A. Stankovic, K. Ramamrithm, "Hard Real-Time Systems." Computer Society Order Number 819, Computer Society Press of the IEEE, 1730 Mass Avenue N.W., Washington DC 20036-1903