A Practical View of Software Measurement and Implementation Experiences Within Motorola

Michael K. Daskalantonakis, *Member, IEEE*

Abstract— **The purpose of this paper is to describe a practical view of software measurement that formed the basis for a company-wide software metrics initiative within Motorola. A multi-dimensional view of measurement is provided by identifying different dimensions (e.g., metric usefulness/utility, metric types or categories, metric audiences, etc.) that were considered in this company-wide metrics implementation process. The definitions of the common set of Motorola software metrics, as well as the charts used for presenting these metrics, are included. The metrics were derived using the Goal/Question/Metric approach to measurement. The paper distinguishes between the use of metrics for process improvement over time across projects and the use of metrics for in-process project control. Important experiences in implementing the software metrics initiative within Motorola are also included.**

Index Terms— **Implementation experience, in-process project control, multidimensional view of measurement, practical view of measurement, process improvement, software metrics, software metrics infrastructure.**

I. Introduction

THE PURPOSE of this paper is to describe a practical view of software measurement, that formed the basis for a company-wide software metrics initiative within Motorola. A *software metric* is defined as a method of quantitatively determining the extent to which a software process, product, or project posesses a certain attribute. This includes not only the formula used for determining a metric value, but also the chart used for presenting metric values, as well as the guidelines for using and interpreting this chart (and metric) in the context of specific projects.

To be practical, software metrics must be defined with their intended use in mind. Goal-oriented measurement (i.e., the identification of measurement goals and important characteristics to be measured before defining the metrics) [1]–[3] ensures such practicality because it provides not only metric definitions, but also the context for making interpretations of their values, so that engineers and managers are able to use them for making decisions.

Several companies are beginning to realize the important role that software metrics can play in planning and controlling software projects, as well as improving software processes, products, and projects over time. Such improvement results in increased productivity and quality, and reduced cycle time, all of which make a company competitive in the software business. Although there are many examples of companies beginning to use metrics in industry (e.g., [4]–[6]), some are finding it a complex and difficult undertaking.

Manuscript received October 1, 1991; revised August 1, 1992. Recommended by R. Selby and K. Torii.

The author is with the Cellular Infrastructure Group, Motorola, Arlington Heights, IL 60004.

IEEE Log Number 9203769.

Results of an industry survey sponsored by Xerox and Software Quality Engineering [7] indicate that fewer than 10% classified it as positive and enthusiastic. In another survey conducted by Howard Rubin and Associates (mentioned in [8]), it is reported that two out of three measurement efforts started, failed, or discontinued after two years. These experiences indicate that the implementation of software metrics is a very complex issue that involves several dimensions. All these dimensions must be addressed through a practical view of measurement in order to increase the likelihood of successfully implementing software metrics within a company or project.

Section II of this paper provides a multidimensional view of software measurement and identifies different ways that software metrics can be used within software projects and a company. Section III discusses the use of metrics for process improvement over time, as well as the use of metrics for in-process project control. Section IV provides the author's experiences from implementing a software merics initiative within Motorola, in terms of the obstacles that were present and how they were addressed, the cost involved in implementing metrics, as well as the benefits obtained so far and expected over the next years. The conclusion is in Section V.

II. A Multidimensional View of Metrics

The purpose of this section is to describe the prerequisites for successful metrics implementation and introduce the dimensions that must be considered when designing and implementing a successful metrics initiative. They should be considered by the function tasked to implement such an initiative.

It is important to understand that the likelihood of a successful metrics implementation increases significantly if several prerequisites are satisfied. These prerequisites specify [9] that the following (preferably automated) systems must be in place: a cost accounting system; a software configuration management system; and a problem reporting/corrective action system. These systems are considered prerequisites for metrics implementation because their existence greatly facilitates any metric data collection and analysis process. If these systems are not in place, the software organization has higher priority items that should be addressed before fully implementing metrics (although a scaled-down metrics initiative may be possible).

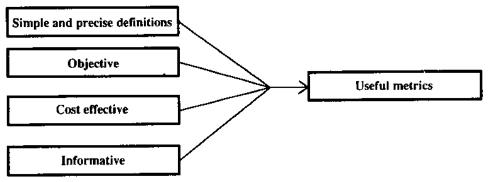


Fig. 1. Sample characteristics of useful metrics.

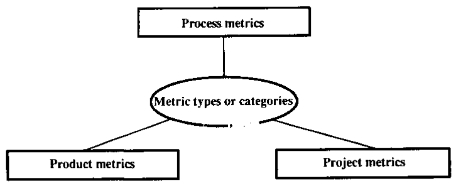


Fig. 2. An example of how software metrics can be classified (based on their intended use).

Also, since metrics are usually used in conjunction with a process that must be controlled and improved, documented processes that are in use should be in place before starting to define process control and improvement metrics. However, some metrics can be defined to be independent of the process used and in that case, there is no need to wait for a process to be defined. There are cases where the existence of metrics provides additional motivation to engineers and managers to go back and define the software processes used in order to improve them over time through the use of metrics.

The dimensions that must be considered when implementing a metrics initiative include the metric usefulness/utility, metric types or categories, metric audiences or users, metric user needs, and the levels of metric application. These dimensions are further described in the following subsections.

2.1. Metric Usefulness/Utility

There are several important characteristics that are associated with useful software metrics. Software metrics must be (Fig. 1):

simple to understand and precisely defined in order to facilitate consistency both in the calculation and the analysis of metric values;

objective (as much as possible) in order to decrease the influence of personal judgement to the calculation and analysis of metric values;

cost effective in order to have a positive return on investment (the value of the information obtained must exceed the cost of collecting the data, calculating the metric, and analyzing its values); and

informative in order to ensure that changes to metric values have meaningful interpretations (e.g., the fact that the estimation accuracy of project effort increased should imply that a better estimation technique was used).

2.2. Metric Types or Categories

Software metrics can be classified under different categories, although it is not unusual that the same metrics belong to more than one category. A classification of metrics based on their intended use follows (Fig. 2).

Process metrics are those that can be used for improving the software development and maintenance process. Examples of such metrics include the defect containment effectiveness associated with defect containment process (e.g., inspection and testing), the efficiency of such processes, and their cost.

Product metrics are those that can be used for improving the software product. Examples of such metrics include the complexity of the design, the size of the source code, and the usability of the documentation produced.

Project metrics are those that can be used for tracking and improving a project. Examples of such metrics include the number of software developers, the effort allocation per phase of the project and the amount of design reuse achieved by the project.

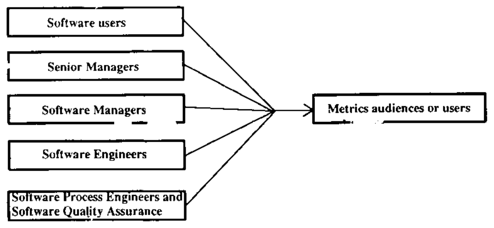

Software users
Senior Managers
Software Managers
Software Engineers
Software Process Engineers and Software Quality Assurance
Metrics audiences or users

Fig. 3. Sample software metric audiences.

2.3. Metric Audiences or Users

There are different potential audiences of software metrics and their primary interests in using metrics are also different (Fig. 3).

Software users are interested in the quality and value of the software products.

Senior Managers are interested in overall control and improvement across projects in the business unit.

Software Managers are interested in control and improvement of the specific software projects they manage.

Software Engineers are interested in control and improvement of the specific software project activities and work products in which they are involved.

Software Process Engineers and Software Quality Assurance are interested in a cross-section of what the four previous audiences are interested in (depending on whether they are working at the business unit level, or the project level).

2.4. Metric User Needs

A software metrics initiative must address the needs of all these potential metric audiences and users by (Fig. 4):

defining metrics and obtaining **consensus/acceptance** by the user community;

training metric users and providing **consulting** support for implementation; and

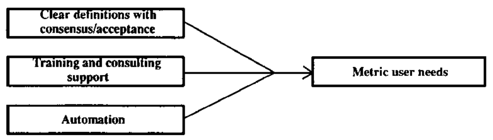

Clear definitions with consensus/acceptance
Training and consulting support
Automation
Metric user needs

Fig. 4. Important metric user needs that must be addressed.

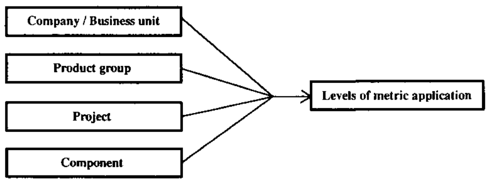


Fig. 5. Sample levels of metric application.

automating the data collection, analysis, and feedback process.

2.5. Levels of Metric Application

Such an initiative must also account for the different levels of measurement. Examples of such levels include (but are not limited to) (Fig. 5) the following.

The **company (or business unit) level,** at which data across several projects may be lumped together to provide a view of attributes such as productivity, quality, and cycle-time across projects.

The **product group level,** at which data across several projects in the same product area may be lumped together to provide a view of the same attributes, but within the product group.

The **project level,** at which data within the project is tracked and analyzed both in-process and post mortem in order to plan and control the project, as well as to improve similar projects across the business unit.

The **component level,** at which data within a component (e.g., subsystem) of a project is tracked and analyzed for managing the development of that component, as well as improving its quality over time.

This section implies that a software metrics initiative must be designed and implemented according to all of these levels and dimensions.

III. Software Metrics Initiative in Motorola

There are several reasons for starting Motorola's company-wide software metrics initiative. Studies published indicated to project participants and management, the usefulness of metrics in improving software engineering and management practices [21], [22]. Engineers and managers wanted to better understand the software development process and be able to determine necessary changes for productivity, quality, and cycle time improvement. They realized that this can be accomplished by measuring both the software development process and product, analyzing the metrics data collected, and determining necessary changes that can lead to more sophisticated techniques for developing and maintaining software. Software managers and engineers have used software metrics as a method for determining progress towards quantified improvement goals, such as Motorola's six sigma quality goal of having no more than 3.4 defects per million of output units from a project.

It was apparent from the start of the metrics initiative (based also on observations from other companies) that software metrics implementation is a complex issue. There are several cultural and human issues that need to be addressed up front in order to assure the success of such an initiative. There is also a need to define and use software processes, focus on continuous process and product improvement, set quantitative goals, and measure the extent to which these goals are achieved. This implies software engineer and manager discipline, as well as acceptance of software measurement as an integral part of the software development process. It also implies the creation of an improvement mentality allowing presentation of data that may indicate significant problems for a project, and concentration on process improvement instead of evaluation of developers.

In addition to these issues, there were also several technical issues that needed to be addressed, such as the lack of common software metrics that are well defined, guidelines for data collection and interpretation, and tools for automating the use of metrics. A Metrics Working Group (MWG) with participation across Motorola's business units was established whose primary purpose was to define a minimal set of software metrics to be used company wide for measuring and eventually improving the quality of the developed software.

The Metrics Working Group has worked for three years intensively to define a common set of software metrics, and support the process of deploying the metrics within software development groups. This common set of metrics is defined in Section 3.1 and their use for tracking improvement over time across projects is explained. Additional processes and metrics were also defined by the Metrics Working Group (or adapted from already existing best practices in industry), so that software metrics are integrated within the process that they attempt to improve. Examples of such metrics associated with specific phases of the software development life cycle where they can be used are included in Section 3.2 and their use for in-process project control is explained. The Metrics Working Group has also created and deployed (through appropriate packaging with training and consulting activities) process and metric definitions for the formal software review and the software test processes using the Goal/Question/Metric approach [1]–[3].

In addition to process improvement over time and in-process project control, software metrics can also be used for measuring customer satisfaction and feeding this information to product designers, analyzing software performance and improving it during software design and development, benchmarking software development practices and results across companies, business units, and projects, etc. However, this paper focuses only on process improvement and in project control.

3.1. Use of Metrics for Process Improvement over Time

The overall philosophy of the company-wide software metrics initiative in Motorola has been: *Measurement is not the goal. The goal is improvement through measurement, analysis, and feedback.* This implies that quantitative improvement goals are identified and metrics data is not only collected, but it is also analyzed for providing improvement feedback to software engineers and managers. Senior management has supported this objective and has made software measurement a required practice by software development projects, as specified in the Quality Policy for Software Development (QPSD). This policy requires the use of metrics by software projects in the following measurement areas:

- delivered defects and delivered defects per size;
- total effectiveness throughout the process;
- adherence to schedule;
- estimation accuracy;
- number of open customer problems;
- time that problems remain open;
- cost of nonconformance;
- software reliability.

After continuous debate within the Metrics Working Group that lasted for about a year, the representatives of different business units agreed on a common set of metrics that addressed the measurement areas identified above. Software productivity was identified as an additional measurement area, and metrics were defined to address this area as well. The primary audience selected for these metrics was senior management, and the metrics were intended to be used primarily for tracking improvement over time, across projects. Although the metrics defined were not perfect, it was decided that it is better to start from a set of metrics addressing the measurement/improvement areas identified, and improve these metrics over time, instead of debating forever, trying to find perfect metrics.

The following definitions of terms are used for describing the metrics defined below.

Software problem: A discrepancy between a deliverable product of a phase of software development and its documentation, or the product of an earlier phase, or the user requirements. The *problem status* can be "open" (the problem has been reported), "closed available" (a tested fix is available to the customer), or "closed" (a tested fix has been installed at the customer site).

Error: A problem found during the review of the phase where it was introduced.

Defect: A problem found later than the review of the phase where it was introduced.

Fault: Both errors and defects are considered faults.

Failure: The inability of a functional software unit to perform its required function. A failure is caused by a defect encountered during software execution (i.e., testing and operation). Problem reports are created as a result of observing a software failure and when analyzed could result in identifying the defect(s) causing the failure.

Released software: Software that has entered the phase of beta test and operation.

Line of code: A physical source line of code, excluding lines that contain only comments or blanks. Lines of code are units of source size. There are two different counts of source size (total and delta). Total source size is the total size of the released software. Delta source size is the size of the source code added, deleted, and modified from the previous software release.

By mapping the measurement areas identified in the QPSD to improvement goals, and mapping these goals to corresponding characteristics (stated in the form of questions), and metrics, a simplified Goal/Question/Metric structure [1] was developed. The goals identified were:

Goal 1: Improve project planning.
Goal 2: Increase defect containment.
Goal 3: Increase software reliability.
Goal 4: Decrease software defect density.
Goal 5: Improve customer service.
Goal 6: Reduce the cost of nonconformance.
Goal 7: Increase software productivity.

The questions and metrics defined were based on practical considerations, such as the scope of the collected data (data primarily from the development organization), and the control over the reported problems (engineering perspective versus customer). Software projects were encouraged to broaden these definitions to take an end-to-end, customer view, and the metrics evolved in this direction over time. Following are the definitions of the common Motorola software metrics (in the context of the goals and questions), as well as a reference to the corresponding chart in Fig. 6 which is used for presenting the metric values. The charts in Fig. 6 are known within Motorola as "the 10–up software metrics charts." *Any data presented in charts within this paper are only for the purpose of providing examples, and should not be considered as representing actual project data.*

Goal 1: Improve Project Planning

Question 1.1: What was the accuracy of estimating the actual value of project schedule?

Metric 1.1: Schedule Estimation Accuracy (SEA) (chart 9)

$$\text{SEA} = \frac{\text{actual project duration}}{\text{estimated project duration}}.$$

Question 1.2: What was the accuracy of estimating the actual value of project effort?

Metric 1.2: Effort Estimation Accuracy (EEA) (chart 9)

$$\text{EEA} = \frac{\text{actual project effort}}{\text{estimated project effort}}.$$

Goal 2: Increase Defect Containment

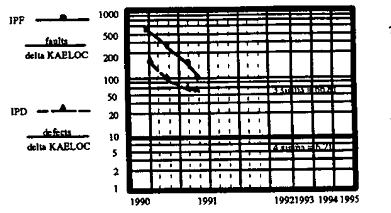
Question 2.1: What is the currently known effectiveness of the defect detection process prior to release?

Metric 2.1: Total Defect Containment Effectiveness (TDCE) (chart 7)

$$\text{TDCE} = \frac{\text{number of pre-release defects}}{\text{number of pre-release defects} + \text{number of post-release defects}}.$$

CHART 1

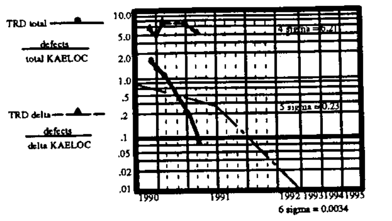
SOFTWARE DEVELOPMENT PROCESS QUALITY (IN SIGMA)



In-Process Faults (IPF) metric and In-Process Defects (IPD) metric for the software released by a Motorola Division.

CHART 2

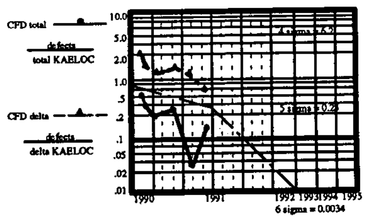
RELEASED SOFTWARE QUALITY (IN SIGMA)



Total Released Defects (TRD) in the software of a Motorola Division.

CHART 3

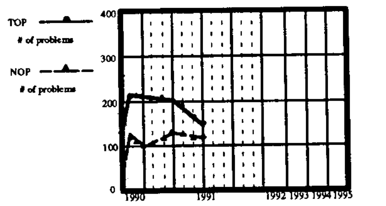
CUSTOMER-FOUND DEFECTS (IN SIGMA)



Customer-Found Defects (CFD) in the software of a Motorola division.

CHART 4

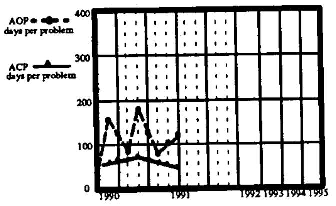
POST-RELEASE PROBLEM REPORT ACTIVITY



Mean New Open Problems (NOP) and Total Open Problems (TOP) for the software of a Motorola Division.

CHART 5

POST-RELEASE PROBLEM REPORT AGING



Mean Age of Open Problems (AOP) and mean Age of Closed Problems (ACP) for the software of a Motorola Division.

(a)

Fig. 6. (a) Sample Motorola software metrics charts.

Question 2.2: What is the currently known containment effectiveness of faults introduced during each constructive phase of software development for a particular software product?

Metric 2.2: Phase Containment Effectiveness for phase i (PCEi) (chart 8)

$$\text{PCEi} = \frac{\text{number of phase i errors}}{\text{number of phase i errors} + \text{number of phase i defects}}.$$

Goal 3: Increase Software Reliability

Question 3.1: What is the rate of software failures, and how does it change over time?

Metric 3.1: Failure Rate (FR) (used at the project level only; not part of the 10–up charts)

$$\text{FR} = \frac{\text{number of failures}}{\text{execution time}}.$$

Goal 4: Decrease Software Defect Density

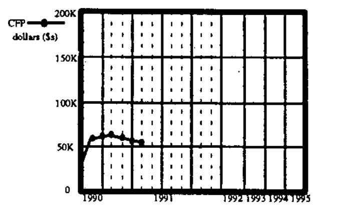
Question 4.1: What is the normalized number of in-process faults, and how does it compare with the number of in-process defects?

Metric 4.1a: In-Process Faults (IPF) (chart 1)

$$\text{IPF} = \frac{\text{in-process faults caused by incremental software development}}{\text{assembly-equivalent delta source size}}.$$

CHART 6

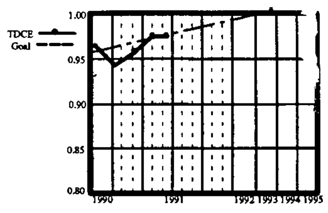
COST TO FIX POST-RELEASE PROBLEMS



Cost to Fix post-release Problems (CFP) within a Motorola Division.

CHART 7

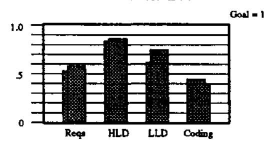
TOTAL DEFECT CONTAINMENT EFFECTIVENESS



Total Defect Containment Effectiveness (TDCE) for the projects of a Motorola division.

CHART 8

PHASE CONTAINMENT EFFECTIVENESS

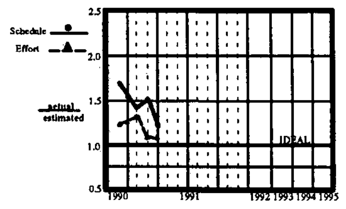


Phase Containment Effectiveness (PCE) for the constructive phases of projects released in the last twelve months within a Motorola Division.

Right: projects released in the last six months.
Left: projects released the prior six months.

CHART 9

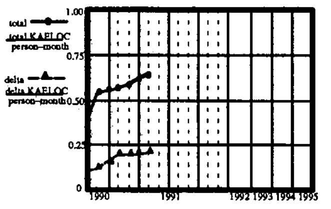
ESTIMATION ACCURACY



Schedule and Effort Estimation Accuracy (SEA and EEA) for the initial estimates of a Motorola Division's software projects.

CHART 10

SOFTWARE PRODUCTIVITY



Software Productivity (SP) total and delta for the projects of a Motorola Division.

(b)

Fig. 6. (b) Sample Motorola software metrics charts.

Metric 4.1b: In-Process Defects (IPD) (chart 1)

$$\text{IPD} = \frac{\text{in-process defects caused by incremental software development}}{\text{assembly-equivalent delta source size}}.$$

Question 4.2: What is the currently known defect content of software delivered to customers, normalized by assembly-equivalent source size?

Metric 4.2a: Total Released Defects total (TRD total) (chart 2)

$$\text{TRDtotal} = \frac{\text{number of released defects}}{\text{assembly-equivalent total source size}}.$$

Metric 4.2b: Total Released Defects delta (TRD delta) (chart 2)

$$\text{TRDtotal} = \frac{\text{number of released defects caused by incremental software development}}{\text{assembly-equivalent total source size}}.$$

Question 4.3: What is the currently known customer-found defect content of software delivered to customers, normalized by assembly-equivalent source size?

Metric 4.3a: Customer-Found Defects total (CFD total) (chart 3)

$$\text{CFDtotal} = \frac{\text{number of customer-found defects}}{\text{assembly-equivalent total source size}}.$$

Metric 4.3b: Customer-Found Defects delta(CFD delta) (chart 3)

$$\text{CFDdelta} = \frac{\text{number of customer-found defects caused by incremental software development}}{\text{assembly-equivalent total source size}}.$$

Goal 5: Improve Customer Service

Question 5.1: What is the number of new problems that were opened during the month?

Metric 5.1: New Open Problems (NOP) (chart 4)

$$\text{NOP} = \text{total new post-release problems opened during the month.}$$

Question 5.2: What is the total number of open problems at the end of the month?

Metric 5.2: Total Open Problems (TOP) (chart 4)

$$\text{TOP} = \text{total number of post-release problems that remain open at the end of the month.}$$

Question 5.3: What is the mean age of open problems at the end of the month?

Metric 5.3: (Mean) Age of Open Problems (AOP) (chart 5)

$$\text{AOP} = (\text{total time post-release problems remaining open at the end of the month have been open})/(\text{number of open post-release problems remaining open at the end of the month}).$$

Question 5.4: What is the mean age of the problems that were closed during the month?

Metric 5.4: (Mean) Age of Closed Problems (ACP) (chart 5)

$$\text{ACP} = \frac{\text{total time post-release problem closed within the month were open}}{\text{number of post-release problems closed within the month}}.$$

Goal 6: Reduce the Cost of Nonconformance

Question 6.1: What was the cost to fix post-release problems during the month?

Metric 6.1: Cost of Fixing Problems (CFP) (chart 6)

$$\text{CFP} = \text{dollar cost associated with fixing post-release problems within the month.}$$

Goal 7: Increase Software Productivity

Question 7.1: What was the productivity of software development projects (based on source size)?

Metric 7.1a: Software Productivity total (SP total) (chart 10)

$$\text{SPtotal} = \frac{\text{assembly-equivalent total source size}}{\text{software development effort}}.$$

Metric 7.1b: Software Productivity delta (SP delta) (chart 10)

$$\text{SPdelta} = \frac{\text{assembly-equivalent total source size}}{\text{software development effort}}.$$

These metrics are applicable primarily at the product group and business unit level and selected subsets are reported monthly (as part of quality reviews and reports) to corporate and senior management. The quality reports are created by the Software Quality Assurance groups within business units based on data collected from projects within these business units. The set of common Motorola software metrics has evolved over time, as feedback was obtained from metrics users. A metrics reference document defining these common metrics, including concrete guidelines for their interpretation and usage has been formally accepted by company-wide committees and is being deployed across the company, with many projects already using these metrics. This document is accompanied by an executive summary for a brief overview of the metrics.

After defining the common set of metrics, overall quantitative quality improvement goals for software processes and products were established. These goals have been stated using Motorola's six sigma quality concept as applied to software. Charts 1 and 2 of the 10–up charts are used for tracking these process and product quality goals. In addition to these goals, software projects and business units were encouraged to define their own improvement goals using the rest of the defined common metrics, based on where their current baseline (i.e., the range of achieved values) is, with respect to these metrics.

In addition to reporting the common software metrics, individual software projects conduct further analysis of metrics data for identifying areas for improvement. Defect data has been found quite useful in this process, because if classified by project phase introduced and by cause, it can lead to actions resulting in significant process and product improvements [13]. Estimation accuracy metrics have helped several projects to define and improve the techniques used for estimating software project schedule, effort, and quality. Software problem-related metrics have also helped projects to track the responsiveness to the needs of those reporting them, and make informed decisions about allocating resources for fixing them versus resources for new software development.

3.2. *Use of Metrics for In-Process Project Control*

The purpose of this section is to provide examples of how software metrics can be used in-process for project control. In-process project control is defined as the ability of software engineers and managers to make informed decisions regarding the current and projected status of a project and take corrective action if necessary. Much of the data collected for the purpose of reporting the 10-up software metric charts, described in the previous section for process improvement, can also be used while the project is still in progress (using different charts) in order to control that project. However, additional refined data is necessary in order to use metrics for in-process project control. Representative examples (but not an exhaustive list) of such use available from industry and piloted/used by projects

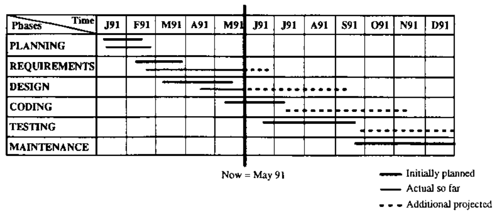


Fig. 7. A chart indicating the progress of planned activities for a project.

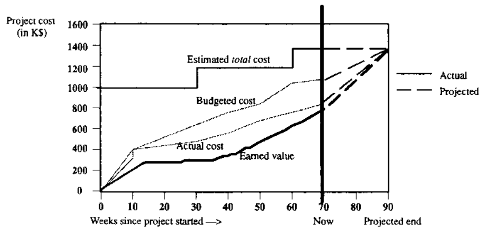


Fig. 8. A chart for tracking the earned value of the project (relative to the budgeted and actual cost).

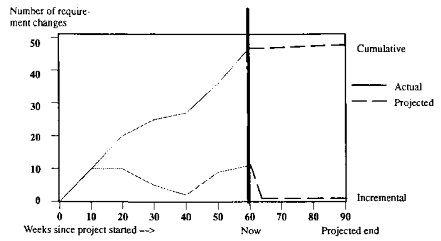


Fig. 9. Number of changes to the requirements document over time for a project.

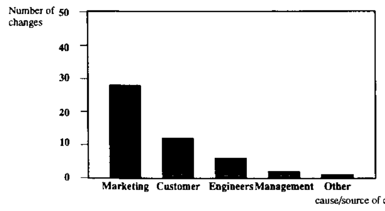


Fig. 10. Pareto chart for identifying major causes/sources of requirement changes in order to take corrective action (in-process).

within the company follow. *The data in the charts are only for providing examples, and should not be considered as actual data from projects.*

3.2.1. Life-Cycle Phase and Schedule Tracking Metric: The purpose of the chart in Fig. 7 is to track the progress of life-cycle phase/schedule progress. It indicates a sample use for tracking the current status of a software project. Management is informed that at the current date (May 1991), the project is supposed to be done with design, and coding should have already started. However, only a portion of the design has been completed so far. The project plan is revised based on the actual data so far, and the additional projected time to complete a phase (or milestone) is plotted. The collection of these charts generated throughout a project can provide also important historical metrics data for software managers. The activities included in the Gantt chart can be presented at a more refined level, depending upon the audience of the chart.

3.2.2. Cost/Earned Value Tracking Metric: The purpose of the chart in Fig. 8 [14] is to allow a manager to track in-process the following cost-related quantities (and update the project plan as necessary).

1) Estimated total cost of the project (estimated at 1000 K$ initially, revised to 1200 K$ at week 30, and revised again to 1350 K$ at week 60).
2) Budgeted cumulative cost of the project.
3) Actual cumulative cost of the project.
4) Earned value of the project (the sum of the budgeted cost for the activities already completed by the project). This value is a good indicator of the current project status.

This metric is important because it summarizes the actual progress of the project (what portion has been completed) and how that relates to the project budget/cost. Tracking these quantities allows the software manager to make informed decisions regarding the progress and viability of the project.

3.2.3. Requirements Tracking Metric: The purpose of the chart in Fig. 9 is to track in-process at the project level requirement changes and determine their impact on the project. Such requirement changes include addition of omitted requirements, and fixes of incorrectly captured requirements. Although enhancements to the software functionality that were not part of the initial scope of the project could also be considered as requirement changes, these changes should be tracked separately using another chart similar to Fig. 9.

If the manager using Fig. 9 determines that there is an unusually high number of requirement changes in the early stages of the requirement phase, the major cause of such changes can be determined in-process. This will allow corrective action to be taken, so that the changes to the requirements are minimized throughout the remaining part of the project. Fig. 10 is an example of a Pareto chart that can be used for this purpose. If marketing is identified as a major cause of requirement changes for adding omitted requirements or fixing incorrect requirements, the manager can enhance the interface with marketing and customers, and possibly use prototyping for ensuring that the required software functionality is understood and captured correctly.

3.2.4. Design Tracking Metric: Software managers are also interested in tracking the progress of designers in designing the software product. Fig. 11 can be used for this purpose. It tracks the cumulative number of requirements traceable into the design, over time. The assumption is that individual requirements are named so that they can be referenced. Another assumption is that a traceability matrix is used to indicate what requirements have already been addressed in the design created so far.

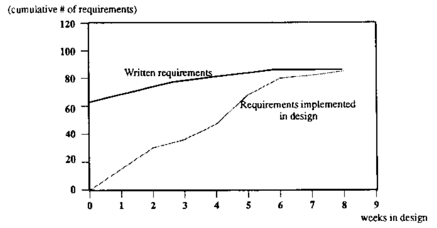


Fig. 11. Design creation progress metric for a software project.

The chart is used while the design is still under development for determining how complete the design is, and how fast it is created (design cycle-time). If the activity of design creation does not progress as fast as initially expected, the software manager determines the impact on schedule of subsequent phases, and the project plan is updated. Ideas on measuring software design complexity are included in [10] and [19].

3.2.5. Fault-Type Tracking Metric: The fault type tracking metric is used for analyzing code faults found with the objective to prevent the introduction of additional code faults. Once software coding has started and some of the modules have been implemented and are being reviewed and unit tested, code fault data starts to accumulate. This data can be analyzed in-process, while coding for additional modules progresses, in order to provide feedback to the coders of such modules regarding types of faults they should avoid introducing. Suppose that a software project uses Ada as the programming language, and the following cause categories for classifying code faults.

C1	— Incorrect or missing initialization of a variable.
C2	— Incorrect interface; call of an operation with the wrong parameters.
C3	— Logic problem, the control flow is wrong, the computation of a value is wrong.
C4	— Error handling problem, exception handled incorrectly, the operation has no recovery mechanism when an incorrect input is encountered.
C5	— The definition of a variable is incorrect, the fields of records are incorrectly defined
C6	— Other.

Fig. 12 is an example of a Pareto chart that can be created using these cause categories to classify code faults. The project participants identify C3 as the most frequently occurring type of code fault (so far), followed by C1. The recommendation is given to the coders working on the rest of the software modules to pay additional attention to the correctness of control flow, computation of values, and explicit initialization of all variables used within their modules. Tools can also be investigated that support the coders in their task by providing a graphical view of their modules, or prompting the coder for initializing any variable which is used without having been initialized yet.

Many additional metrics can be used in the coding phase for in-process project control (several of them automated through commercially available tools). For example, modules with high cyclomatic complexity [15] can be identified, and the manager can allocate additional resources for reviewing and unit testing these modules, or even rewrite them if they are unnecessarily complex.

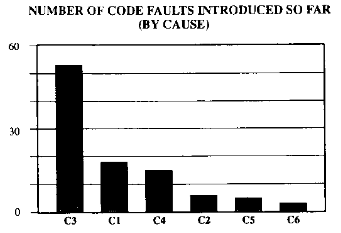


Fig. 12. Pareto chart for identifying what type of code faults is occurring most frequently, so that coders of additional modules can avoid their introduction in such modules.

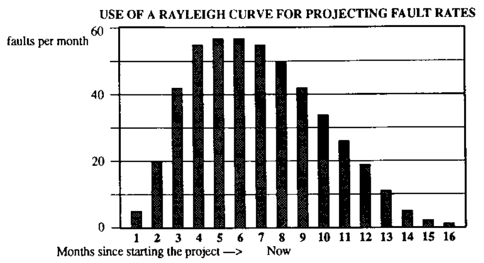


Fig. 13. Fault finding rate over the life-cycle of a software project.

3.2.6. Remaining Defects Metric: The chart in Fig. 13 [16] can be used in conjunction with a technique for estimating the number of faults remaining in a software project. This technique assumes that the fault finding rate for a software project has the shape of a Rayleigh curve that can be represented in the form of an equation. This equation provides the number of faults per month as a function of several known (or estimated) parameters from the project. A key parameter that must be estimated (based on historical fault data) is the total number of faults expected to be found over the life-cycle of the software product.

In the example of Fig. 13, if the software manager has data from the first months into the project, the Rayleight curve (and its corresponding equation) can be used for the purpose of projecting the fault finding rate over the remaining months of the software life-cycle. This fault finding rate can be used to project the number of faults to be found within a specified number of months. In this example, month 8 may correspond to the time that integration testing starts for the software project. Using the data so far (bars 1–7), and through fitting it in a Rayleigh curve, the projected remaining number of defects to be found in the software project can be estimated (based on the number for months 8–16).

3.2.7. Review Effectiveness Metric: The review effectiveness metric can be used to track review effectiveness and improve reviews and product quality over time. A control chart

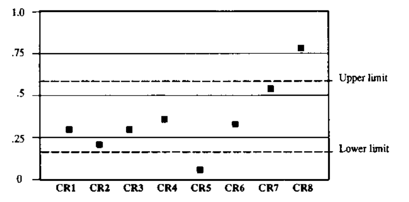


Fig. 14. Error density of source code reviewed within the project (or similar projects in the Division).

can be used in-process for identifying any potential problems with the review (inspection) process or the product reviewed. Fig. 14 is an example of such a chart, indicating the error density of source code reviewed so far within the project (or within similar projects in the Division) [17]. CRi indicates code review i. By using this chart in-process, the software manager determines that the review process used in CR5 must be examined further (by talking to the reviewers participating in the process). If the review process steps were followed, and the right people participated, the data for CR5 indicates that the quality of the code reviewed was relatively high. If the review process steps were not followed, and the right people did not participate as reviewers, it is an indicator that additional errors may exist in the code, and an additional review may be necessary.

By examining the same chart and attributes (in-process) for CR8, the software manager can make the following decisions. If the review process steps were not followed or the right people were not among the reviewers, an additional review and/or rewrite may be necessary for the code reviewed. However, if the process steps were followed and the right people were among the reviewers, most of the errors are likely to have been found. If the code reviewed was critical for the whole software product functionality, then a rewrite may be necessary (due to the high error density; fixes to errors found may have introduced additional faults in the code).

3.2.8. Problem Severity/Priority Tracking Metric: The problem severity/priority tracking metrics are used to track progress of fixing defects described in problem reports found in testing and make software release decisions. Total Open Problems are tracked overtime for problems of different severities (and/or priorities), so that the manager is able to determine whether or not major problems are being fixed, and whether or not the software project is getting closer to releasing the product, based on the release criteria used. An example of such release criteria for a large project may be that at release time all severity 1 (i.e., crash causing) and 2 (i.e., major functionality affected) problems found so far should be closed. Also, that no more than five problems of severity 3 (i.e., minor functionality affected) should exist, and no more than 15 problems of severity 4 (i.e., cosmetic) should exist in the software, all these problems will be communicated to the customers. Fig. 15 can then be used for making release decisions, in addition to tracking the progress of problem fixing activities (the three bars per severity correspond to the last three weeks before the data is reported).

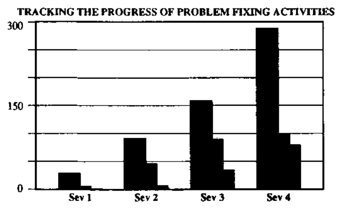


Fig. 15. Total open problems for the past three weeks of system testing for a software project.

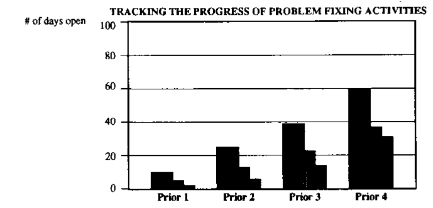


Fig. 16. Age of Open Problems for the past three months of system testing for a software project.

Fig. 16 can also be used by the software manager in-process for tracking the Age of Open Problems metric by priority level, prioritizing resource allocation for problem fixing activities. If the age of priority 1 open problems is high, or is not dropping significantly over time, testing is disrupted (because the software crashes and tests cannot be executed). The manager can allocate the necessary resources for fixing such problems and is able to visualize what the impact is by examining Fig. 16.

An additional use of severity (and/or priority) data associated with problem reports and defects found in testing is to create a list of severity 1 and 2 defects found so far, and have a meeting with the designers and coders in order to identify where else in the software these same defects may exist. The output of this meeting can be used for fixing additional defects identified through this process, without waiting for testing (or the customers) to find them first.

In addition to the techniques discussed in this section, the Failure Rate metric can be used (in conjunction with a software reliability model [18]) for determining the current reliability level of the software and making release decisions. Also [20] provides additional examples of tracking quality-related attributes from a Software Quality Assurance perspective.

IV. Experiences from Implementing Motorola's Software Metrics Initiave

4.1. Software Metrics Infrastructure

Implementing a company-wide software metrics initiative can be a long term process. However, the benefits obtained for the company are certainly worth this effort. It was understood early in the process of implementing this initiative that there is a need for a *software metrics infrastructure.* A software metrics infrastructure is what needs to be put in place for he purpose of facilitating metrics implementation within business units. This consists of working groups with participation across the company (e.g., the Metrics Working Group), the deliverables, training workshops on metrics, tools automating metrics, consulting support for metric implementation within projects, etc. These items are further explained below.

In addition to the Metrics Working Group, a Metrics Users Group (MUG) has been established for sharing user experiences of implementing software metrics in projects. The Metrics Users Group has representation across business units, meets quarterly, shares experiences regarding the use of tools to automate metrics, and organizes demos of such tools. The group is also involved in organizing an Annual Software Metrics Symposium within the company.

Additional activities and outputs that are part of the software metrics infrastructure established by the Metrics Working Group follow.

- Clarifications of metric definition, interpretation, and use are provided by the Metrics Working Group during its regular meetings (the group meets twice a quarter since 1988), and they are "packaged" as part of the software metric documentation and training material for dissemination to metric users.
- In order to address the "training" and "consulting support" dimensions of metrics implementation, a two-day training workshop on software metrics has been developed and has been taught across the company in the past two years. Hands-on consulting activities by the instructor follow the training sessions if requested by the workshop participants. This has been found a very effective mechanism of software technology transfer.
- In order to address the "automation" dimension of metrics implementation, requirements for an automated metrics data collection, analysis, and feedback system were also created by the Metrics Working Group and were provided to tools groups involved in automating software metrics.
- Criteria for evaluating metrics tracking systems were also created for facilitating the process of selecting commercially available metric tools. A list of metric tools that exist within the company or are available form industry was created and disseminated to interested metric users.
- Support for further analysis of collected metric data has been provided to metric users through generic defect classification schemes and examples of using these schemes for creating process improvement recommendations.
- The Defect Prevention Process [13] has been recognized as an effective mechanism to ensure that process improvement is achieved through defect data analysis. This approach has been championed by the MWG and several projects have started using this approach.
- Guidelines to interested business units for creating a function responsible for implementing software metrics are also available and used. As any other activity, if no function is identified with the task of implementing metrics it is almost certain that it will not happen.
- A method for assessing software measurement technology [11] has been created and it is used for providing feedback to projects about priority items that will further support metrics implementation.
- Customer satisfaction measurement through surveys (from a software perspective) is also encouraged through output of the Metrics Working Group.

4.2. Additional Implementation Experiences

An important lesson learned from the application of software measurement is that it is better to start from a set of metrics addressing important improvement areas, and evolve these metrics over time, instead of debating forever, trying to find perfect metrics. Another lesson is that as engineers and managers start using metrics, they realize the potential benefits of such use, and they start investigating additional ways to obtain even more benefits. The initial charts defined for presenting the common set of metrics were targeted toward senior management and the data presented was dependent on post-release data (primarily due to the need to minimize the cost associated with data collection). These charts are useful for providing an overview of the software quality status, and senior management has used them to track trends over time, as well as for benchmarking purposes.

Software engineers and managers who started using these charts expanded the use of metrics for in-process project control and feedback. As metrics data is collected within the software development process, the data should be analyzed in order to determine current project status and make projections of estimated status for the next phases of the project. This use within the project provides timely value to the engineers and software managers involved in the project. In addition to the initial 10–up software metric charts, in-process charts were defined and are used within software projects as explained in Section 3.2

In addition to the in-process charts, the Metrics Working Group has created process and metric definitions for the formal software review, and the software test processes. The metrics defined for controlling and improving the software review and test processes were developed through the application of the Goal/Question/Metric approach which was found very useful. The process and metric definitions are appropriately packaged in order to address the needs of different audiences, and training material has been developed and taught for the purpose of transferring this technology and ensuring its use by the product groups. A recent survey of software engineers and managers conducted indicates that a very high percentage (67% of those surveyed) use the software review package which was deployed by the Metrics Working Group over the last three years. Additional process/metrics packages will be investigated for processes such as the software design process and metrics for improving this process.

There have been several requests over time from metrics users across the company for a centralized location where metrics data can be stored and compared to data from other projects. However, the approach taken was that a metrics initiative is more manageable when it is *initiated* by encouraging localized (decentralized) data storage, analysis, and feedback, so that the data is close to its source. It makes more sense to use data when the context for the projects is known (e.g., products within a product group), as opposed to using data from projects in different product groups, projects of different size and complexity, etc. This is the reason why in addition to any metric data collected, data regarding the project from which this data was obtained should also be collected, stored, and used for analysis purposes. Decentralized databases storing data from local projects could be connected once the initiative is well established for providing benchmarking data to interested projects across the company.

A software metrics initiative should emphasize as its initial stages that the engineers and managers involved in the project are the best people to analyze the collected data, because they have expertise in the project domain and can interpret what the data indicates. An external consultant who has expertise in data analysis for process improvement and project control can help project participants to initiate such activities, and document some examples of how to do data analysis. It is best that this consultant be involved early in the project so that useful metrics are selected and the mechanisms are put in place to collect valid data and analyze them for improvement. Once this is done for a couple of projects, the project participants should take over, especially since the resources of software metric experts are limited (there is only a limited number of experts in this area that is available to date). The project participants should be able to champion data analysis and feedback once the metrics consultant has initiated these activities within the product group or business unit.

Another commonly found request to be expected when implementing software metrics is to pick only one metric to be used in a project, so that the cost is minimal. However, such use of a metric can be misleading, because it is really a set of metrics that should be tracked and analyzed in order to obtain a more accurate picture of several important attributes regarding a project or organization. If only one metric is used, projects can manage to optimize the values of that metric (indicating positive results). However, these projects may have significant problems that are not obvious by examining a single metric.

There is a cost involved when implementing a software metrics initiative. There are approximately eight participants present in the Metrics Working Group meetings (twice a quarter). There are also about 15 people present in the Metric Users Group meetings (quarterly). These people are involved at least on a part time basis within their organizations in implementing software metrics. Cost is also involved in terms of implementing software metric tools, but it can be minimized by avoiding duplication of effort.

There is an example within a Motorola Division where the resources used were 3 persons per year for approximately 350 software engineers (less than 1% of resources). In an example from another Division, the resources used were 0.75 person-years per year for approximately 70 engineers (about 1% of the resources). Cost associated with post-release metric data collection has been insignificant compared to the benefits. The cost associated with in-process metric data collection can be higher, but such cost can be minimized through automation. In general, the overall cost is acceptable and justified. The benefits obtained so far through quality, productivity, and cycle-time improvement (which are expected to continue in the future), are well worth the investment made.

There are several additional benefits that have been obtained so far from implementing software metrics. People have started thinking more seriously about software process and quality. The data has helped projects understand the extent of the problems they were facing and motivate them to improve. The metrics have helped establish local baselines (i.e., ranges of achieved data values), and focus on actions with quantitative results. There are cases of significant quality and productivity improvements due to implementing several software engineering practices, including metrics.

For example, the focus within a Motorola Division on improving software quality (and tracking results through metrics) has achieved 50X reduction in released software defect density within 3.5 years. However, it is important to understand that presenting the 10–up software metric charts did not improve quality by itself. It is the quality initiative taken as a result of analyzing the data in the charts that made the difference.

There are also many indirect benefits from implementing software metrics, including cases where the use of metrics has helped to improve ship-accpetance criteria, and schedule estimation accuracy. Software development groups are expected to learn from their mistakes from previous projects (through post-mortem analyses) and take action to avoid them. It is also expected that as a result of improving software quality, there will be significant improvement in Total Customer Satisfaction. Another long range benefit expected (which has been actually achieved so far within Motorola) is significant cost reduction due to improved quality. This results from reduced rework and the use of resources for new software development instead of fixing problems. In addition to cost reduction, reduced cycle time is also expected.

Software metrics is only one of the initiatives taken in the area of software quality. Additional initiatives include the use of a Quality System Review for software, Senior Management Forums for software, Software Engineering Institute assessments, software engineering education, technology transfer, and benchmarking/use of best practices. Motorola has been awarded the First Malcolm Baldrige National Quality Award in 1988 for its successful efforts and results on improving quality in all aspects of its business.

V. Conclusion

By addressing the areas discussed in this paper, Motorola has been successful in the implementation of a company-wide software metrics initiative with minimum resources. The level of expertise in using metrics varies across software development projects, but increases over time. However, additional work is necessary for ensuring that the software metrics initiative is institutionalized across all software development projects. Results from the use of metrics (documented in the proceedings of the Annual Software Metrics Symposium) indicate several examples where benefits have been achieved.

It is important to remember that metrics can only show problems and give ideas as to what can be done. It is the actions taken as a result of analyzing the data that bring the results. This is the reason why it is critical for metrics users to understand that measurement is not the goal. The goal is improvement, through measurement, analysis, and feedback.

Acknowledgment

Motorola's Software Process Engineering Group (SPEG) within Software Research and Development (SRD) has championed the use of metrics across the company over the last four years. The work presented in this paper was done while the author was working in that group. The author would like to acknowledge the contribution to this work by many Motorolans who participated in the Software Quality Subcommittee (SQSC) of the Software Engineering Technology Steering Committee, and its Metrics Working Group over time. All these people have been significant contributors ensuring that software metrics are used for driving software process and product improvement over time. The author would like to recognize explicitly the contributions of Dr. R. H. Yacobellis (Manager of the Software Process Engineering Group) in supporting the metrics initiative from a management perspective. Also, Dr. V. R. Basili (Professor of Computer Science, University of Maryland at College Park) who provided consulting support to the Metrics Working Group and ensured a continuous focus on creating output that is of value to its customers (i.e., metric users across Motorola).

References

[1] V.R. Basili and D.M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Trans. Software Eng.*, vol. SE-10, pp. 728–738, 1984.
[2] V.R. Basili and H. D. Rombach, "Tailoring the software process to project goals and environments," in *Proc. Ninth Int. Conf. Software Engineering,* 1987.
[3] ——, "The TAME project: Towards improvement-oriented software environments," *IEEE Trans. Software Eng.,* vol. SE-14, pp. 758-773, June 1988.
[4] S.L. Pfleeger, J.C. Fitzgerald, and A. Porter, "The CONTEL software metrics program," in *Proc. First Int. Conf. Applications of Software Measurement,* Nov. 1990.
[5] F.E. McGarry, "Results of 15 years of measurement in the SEL," in *Proc. Fifteenth Annual Software Engineering Workshop,* NASA/Goddard Space Flight Center, Nov. 1990.
[6] R. B. Grady and D.L. Caswell, *Software Metrics: Establishing a Company-Wide Program.* Englewood Cliffs, NJ: Prentice Hall, 1987.
[7] B. Hetzel, "The software measurement challenge," in *Proc. First Int. Conf. Applications of Software Measurement,* Nov. 1990.
[8] G. Miluk, "Cultural barriers to software measurement," in *Proc. First Int. Conf. Applications of Software Measurement,* Nov. 1990.
[9] F.J. Buckley, "Rapid prototyping a metric program," in *Proc. First Int. Conf. Applications of Software Measurement,* Nov. 1990.
[10] ——, "Rapid prototyping a metrics program," in *Proc. First Int. Conf. Applications of Software Measurement,* Nov. 1990
[11] V.R. Basili and D.H. Hutchens, "An empirical study of a syntactic complexity family," *IEEE Trans. Software Eng.,* vol. SE-9, pp. 664–672, 1983.
[12] M.K. Daskalantonakis, V.R. Basili, and R.H. Yacobellis, "A method for assessing software measurement technology," *Quality Engineering J. American Society for Quality Control,* vol. 3, 1990–91.
[13] C. Jones, *Applied Software Measurement–Assuring Productivity and Quality.* New York: McGraw-Hill, 1991.
[14] R.G. Mays *et al.*, "Experiences with defect prevention," *IBM Syst. J.* vol. 29, 1990.
[15] D. Youll, *Making Software Development Visible.* New York: Wiley Series in Software Engineering Practice, 1990.
[16] S.D. Conte, H.E. Dunsmore, and V.Y. Shen, *Software Engineering Metrics and Models.*Benjamin/Cummings, 1986.
[17] L. Putnam, *Quantitative Software Management (QSM) Approach.* McLean, VA: Quantitative Software Management, 1990.
[18] J. Kelly and J. Sherif, "An analysis of defect densities found during software inspections," Fifteenth Annual Software Engineering Workshop, Goddard Space Flight Center, Greenbelt, Maryland, Nov. 1990.
[19] J.D. Musa, A. Iannino, and K. Okumoto, *Software Reliability–Measurement, Prediction, Application.* New York: McGraw-Hill, 1987.
[20] D. Card and R. Glass, *Measuring Software Design Complexity.* Englewood Cliffs, NJ: Prentice Hall, 1990.
[21] B. Glick, "An SQA quality tracking methodology," in *Proc. Int. Conf. Software Maintenance,*Nov. 1990.
[22] V.R. Basili and R.W. Selby, "Comparing the effectiveness of software testing strategies," Univ. Maryland, College Park, *Tech. Rep. TR-1301,* May 1985.
[23] B. W. Boehm, "Understanding and controlling software costs," *IEEE Trans. Software Eng.,* vol. 14, Oct. 1988.
[24] J. Brian Dreger, *Function Point Analysis.* Englewood Cliffs, NJ: Prentice-Hall, 1989.

Michael K. Daskalantonakis (M'88) received the Bachelor of Science degree in computer engineering from the University of Patras, Patras, Greece, in 1985, and the Master of Science in computer science from the University of Maryland, College Park, in 1988.

He has worked in the past as a computer programmer for IBM Corporation, a research assistant at the University of Patras in the ESPRIT project GRASPIN, and a research assistant at the University of Maryland in the TAME project. He joined Motorola, Inc. in July 1988. He worked within Corporate Software Research and Development in the area of software process engineering and measurement for improvement. He has been the champion of software measurement technology within Motorola. Additional areas of significant contribution include research, training, and consulting in SEI assessments, formal software reviews, Quality System Review, and software testing. He is currently a Lead Engineer in the EMX 2500 product area within Motorola's Cellular Infrastructure Group. He provides hands-on support for process and metrics implementation within software projects. He has published several papers on selected software engineering topics and served as a panelist at metrics conferences.

Mr. Daskalantonakis is a member of the ACM.