UMIACS–TR–89–75
CS–TR–2289

July, 1989

# Integrating Automated Support for a Software Management Cycle into the TAME System†

Toshihiko Sunazuka
NEC Corporation
Tokyo, Japan

Victor R. Basili
Institute for Advanced Computer Studies
Computer Science Department
University of Maryland
College Park, MD 20742

## ABSTRACT

Software managers are interested in the quantitative management of software quality, cost and progress. There have been many of models and tools developed, but they are of limited scope. An integrated software management methodology, which can be applied throughout the software life cycle for any number purposes, is required.

The TAME (Tailoring A Measurement Environment) methodology, developed at the University of Maryland, is based on the improvement paradigm and the Goal/Question/Metric (GQM) paradigm. This methodology helps generate a software engineering process and measurement environment based on the project characteristics.

The SQMAR (Software Quality Measurement and Assurance Technology) developed in NEC is a software quality metric system and methodology applied to the development processes. It is based on the feed forward control principle. Quality target setting is carried out before the Plan–Do–Check–Action activities are performed.

These methodologies are integrated to realize goal–oriented measurement, process control and visual management. The Software Management Cycle is a substantiation of these concepts. Based on the TAME process model, development and management environments can be generated. The SQMAT system helps target setting, data analysis and visual display.

In this paper we discuss a metric setting procedure based on the GQM paradigm, a management system called the Software Management Cycle (SMC), and its application to a case study based on NASA/SEL data. A method for evaluation Software Management Cycle process is described. The expected effects of SMC are quality improvement, managerial cost reduction, accumulation and reuse of experience, and a highly visual management reporting system.



## KEYWORDS

TAME, improvement paradigm, Goal/Question/Metric paradigm, SQMAR, Plan–Do–Check–Action activities, process control, visual management, software engineering process, goal–oriented measurement, software quality metrics.


---


† Research supported in part by NASA grant NSG–5123 and NEC (through the Industrial Associates Program of the Department of Computer Science.)

## 1. Introduction

Management plays a key role in the software development process. In the end, it is management's responsibility to produce and deliver a quality product productively and profitably and to generate corporate credibility with the customer. Thus, effective management methodologies are needed to support management in assessing the current status of the project and achieving delivery of the final system on–time, within budget, and with the specified product qualities. It would also be useful if the methodology supported the improvement of quality and productivity on the current project and on future projects. Many companies are working to provide such methods for their managers.

However, it is difficult to assess the current status of a project precisely because of the lack of visibility of the software during development. It is even more difficult to predict project progress because of the lack of clearly defined goals, the lack of feedback in the achievement of those goals, and the difficulties caused by the variation in personnel.

## 2. Supporting Methodologies

Thus, requirements for the management methodology include the ability to make the software as visible, quantifiable and objective as possible. Several methodologies and paradigms use metrics to satisfy these management needs during development. There have been many software metrics proposed in the literature that attempt to provide the visibility, quantification and objectivity [Boeh76, McRW77, Muri80, BaKa83].

From a customer perspective of product quality, a comprehensive set of quantifiable software characteristics were proposed by Boehm, et al. [Boeh76] and later refined by McCall and Walters [McRW77]. Based on these studies, Software Quality Metrics (SQM) was developed by Murine (METRIQS Incorporated) as a quantitative software quality assessment technology [Muri80].

### SQMAT

Based upon the SQM, the NEC Corporation has developed a Software Quality Measurement and Assurance Technology (SQMAT) [AzSM87, AzSu86, SuAY85] and has been using it as one of the support tools in their software quality control (SWQC) group activities [Mizu82]. Quality control seminars are held periodically for every level of worker; programmer through general manager. The seminars are used to motivate as well as educate everyone with respect to the quality control technologies.

SQMAT is a software quality metric system and methodology applied to the development processes, which takes experimental SQM results into consideration. SQMAT consists of a quality measurement and evaluation method with three levels of quality criteria, and a support tool for a visual display for management. Its most notable feature is that the feed forward control principle is employed in addition to the feedback control principle. That is, quality target setting is carried out before the Plan–Do–Check–Action activities (Deming's PDCA cycle) are performed. SQMAT procedures are defined as follows:

(1) In the TARGET phase, a quality priority ranking is established for the individual quality characteristics, based on the users' requirements and the development policy. It is important to clarify the quality target, i.e., classify the quality characteristics into 3 categories and set the target quantitatively.

(2) In the PLAN phase, Software Quality Measurement Criteria (SQMC), are set up and methods for achieving the target quality are discussed in advance, primarily with the

quality assurance people and managers.

(3) In the DO phase, high quality software is produced by complying with development standards and SQMC as guidelines. Before the formal review, the developer executes a quality self–check.

(4) In the CHECK phase, the software is checked and evaluated against the individual quality criteria set up in the PLAN phase. Quality is measured by a third party. If errors are detected, problem reports are drawn up. After scoring, score sheets and quality graphs are developed, and the achieved quality is judged by comparing it to the target quality level.

(5) In the ACTION phase, corrective action is taken, based on problem reports. Achieving the quality target permits proceeding on to the next phase. SQMAT can be applicable not only to large scale software, but also to small projects. NEC's experience with the approach has had measurable results. For example, based upon comparison with historical data, (1) a number of errors have been eliminated during the design and implementation phases, and (2) productivity (measured by lines–of–source–code/hour) has increased by 10%.

### The Improvement Paradigm

The Quality Improvement Paradigm [Bas85a] for software engineering processes is a top level paradigm that is based upon the scientific method as applied to software evaluation. It provides the view of software evolution as an experimental process from which we must learn and improve the current project as well as future projects (Characterize, Set Goals, Choose Methods, Build, Analyze, Learn and Feed Back). It is a meta–life cycle model that aims at improving the software quality and productivity based upon measurement and reuse of experience. It needs to be instantiated for a variety of sub–activities, e.g. specific processes such as testing, product reviews, managing. It consists of six major steps:

(1) Characterize the current project environment.

(2) Set up goals and refine them into quantifiable questions and metrics for successful project performance and improvement over previous project performances.

(3) Choose the appropriate software project execution model for this project and supporting methods and tools.

(4) Execute the chosen processes and construct the products, collect the prescribed data, validate it, and analyze the data to provide feedback in real–time for corrective action on the current project.

(5) Analyze the data to evaluate the current practices, determine problems, record the findings and make recommendations for improvement for future projects.

(6) Package the experience in the form of updated and refined models and other forms of structured knowledge gained from this and previous projects and proceed to step 1 to start the next project.

This paradigm is aimed at providing a basis for corporate learning and improvement [BaRo87] and is based upon experience with measurement and evaluation of software development in a number of companies.

### Goal Question/Metric Paradigm

The Goal/Question/Metric (GQM) paradigm [BaWe85, BaSe84] is a mechanism for generating measurement in a goal–directed manner. It represents a systematic approach for setting the project goals (tailored to the specific needs of an organization), defining them in an operational, tractable way by refining them into a set of quantifiable questions that in turn imply a specific set of metrics and data for collection (addresses the aspects related to step 2) of the improvement

paradigm). Appropriate metrics are tailored to each project based on the G/Q/M templates and past experience. It includes the development of data collection mechanisms, e.g., forms, automated tools, the collection and validation of data, and the analysis and interpretation of the collected data and computed metrics in the appropriate context of the questions and the original goals.

In order to support the process of setting goals and refining them into quantifiable questions, a set of templates for setting goals, and a set of guidelines for deriving questions and metrics has been developed [BaRo88]. These templates and guidelines reflect our experience from having applied the GQM paradigm in a variety of environments [RoBa87, WeBa84, BaWe81].

Goals are defined in terms of purpose, perspective and environment. Different sets of guidelines exist for defining product–related and process–related questions. Product–related questions are formulated for the purpose of defining the product (e.g., physical attributes, cost, changes and defects, user context), defining the quality perspective of interest (e.g., functionality, reliability, user friendliness), and providing feedback from the particular quality perspective. Process–related questions are formulated for the purpose of defining the process (process conformance, domain conformance), defining the quality perspective of interest (e.g., reduction of defects, cost effectiveness of use), and providing feedback from the particular quality perspective.

The TAME (Tailoring A Measurement Environments) system [BaRo88] is a measurement environment that supports and integrates the Quality Improvement and the Goal Question Metric paradigms.

Based on the work at NEC, the TAME project, and the managerial requirements specified above, a management methodology, called the Software Management Cycle (SMC), has been developed. Its main concepts are goal oriented, process control and visual management. Management procedures, support tools and forms, and an evaluation method are provided as part of SMC.

## 3. Relationship of the SQM, GQM, and SQMAT

The SQM and the GQM are both mechanisms for measuring software quality. Both models are top–down and characterize quality characteristics at three levels. In the SQM, these levels are Factor, Criteria, and Metric. For example, a high level factor such as correctness is defined by the set of criteria traceability, completeness, and consistency which in turn are defined in terms of a predefined set of metrics.

The GQM model consists of a goal, which is specified by a set of quantifiable questions, which in turn are defined by a set of metrics and data distributions tailored to the specific environment. Thus to define a high level goal like correctness of the final product, we must define a set of questions that characterize the product (with respect to its physical attributes, cost of development, changes and defects, and customer base and operational profile), define a model for correctness (which could include such concepts as traceability, completeness, and consistency), provide insights into the validity of the model and the data within the particular environment, and the results of the model along with some possible substantiation of the model results.

The SQM model predates the GQM model, but the latter is more general. The GQM can be used to characterize, evaluate, predict, or motivate a product, process, model or metric, with respect to a variety of perspectives (e.g. customer, developer, user, manager, etc.) based upon an open ended definition of quality. It takes into account the specific environment in which the product has been developed as well as assessment of such things as an evaluation of how well the

particular methods were used, how well the domain of application was understood in order to help interpret the resulting evaluation metrics appropriately. It also involves the feedback of information for future development through learning.

The SQM model is written from the point of view of determining a set of quality characteristics of the final product from the point of view of the customer. It does not measure process for developing that product and since its viewpoint is that of the customer, it provides limited support for learning, feedback and improvement within the development organization. Its measurement process tends to be passive and is not focussed on capturing the causes of the quality problems.

The measurement focus of SQM as used in SQMAT has evolved and widened over time and is currently more consistent with the GQM. This wider view of SQM uses metrics to measure quality of an intermediate product from the point of user, developer and so on.

<table>
<tr><th></th><th colspan="2">GQM</th><th colspan="3">SQM</th></tr>
<tr><th></th><th colspan="2"></th><th colspan="2">narrow-sense</th><th>wide-sense</th></tr>
<tr><td>Objective</td><td colspan="2">Characterize, Assess,<br>Predict, Motivate</td><td colspan="3">Assess (Quality)</td></tr>
<tr><td>Structure</td><td colspan="2">Goal<br>Question<br>Metric</td><td colspan="3">Factor<br>Criteria<br>Metric</td></tr>
<tr><td>Usage</td><td colspan="2">Project & Quality Management</td><td colspan="3">Quality Management</td></tr>
<tr><td>Object</td><td colspan="2">Any Product, Process,<br>Model, or Metric</td><td colspan="2">Product</td><td>Any Product<br>Process</td></tr>
<tr><td>Viewpoint</td><td colspan="2">Developer, User,<br>Manager, Corporate</td><td colspan="2">User</td><td>(same as<br>GQM)</td></tr>
<tr><td rowspan="3">Establish-<br>ment manner<br>of GQM or<br>SQM</td><td>G</td><td>Select or Tailor</td><td>F</td><td colspan="2">Select</td></tr>
<tr><td>Q</td><td>Select or Tailor</td><td>C</td><td colspan="2">Select</td></tr>
<tr><td>M</td><td>Select or Tailor</td><td>M</td><td>Select</td><td>Select<br>or Tailor</td></tr>
</table>

**Table 1. Features of GQM and SQM.**

## 4. Software Management Cycle (SMC)

The Quality Improvement Paradigm provides a top level organizational perspective on the software development and maintenance process. SMC is the management procedure and support system under that paradigm. It emphasizes three concepts; goal–oriented measurement, process control, and visual management. In response to each concept, several activities are necessary. These activities, performed during the management procedure, make it possible for management to achieve higher quality and productivity.

The management procedure used in SMC consists of the following five steps:

(Step 1) Define system/project characteristics

It is important to define the system characteristics in detail to reflect the user requirements for development. A set of system/project characteristics forms are prepared to gather information on the requirements and the current project status.

This is equivalent to the first step of the Quality Improvement Paradigm. The system engineer is responsible for understanding the customer requirements for the particular project correctly. The development environment should be also clarified. This characterization permits the comparison of the current project with prior projects with similar characteristics. This information is used in the next step.

(Step 2) Select Goals,Questions, and Metrics

To achieve high quality and productivity, it is necessary to set the specific objectives. This is the key step to the success of the project. Unless the goals are appropriate, the project will fail. The GQM paradigm is used to do this. It satisfies the requirement for goal–oriented measurement. It helps both developers and managers clarify the objectives of the project prior to development.

Guidelines and templates are used to establish the particular GQM used. Templates from prior systems can be used or modified for this project. For each metric, measurement instructions are prepared, which include the importance of metric, the collection method and person responsible, the data presentation, the decisions affected etc.

Besides the set of goals and metrics for the particular project, a common set of managerial metrics have been specified to be applied to all projects. We can gather the data for getting the level of quality and productivity through projects and development phases. The metrics from this common set are shown below.

[ Quality ]
- Number of detected errors at test phase (from integration test through system test)
- Number of detected errors within six months after release

[ Productivity ]
- Number of specification pages
- Number of non–comment source statement
- Effort at each phase by man–hour

(Step 3) Select activities

Methods to effectively achieve the objective are considered at this time. Appropriate activities for economically producing the software and managing the quality of the project can now be chosen based on the specific objectives laid out in the GQM.

This step is critical to achieve the objectives. Setting goals, without specifying the means to achieve them, is meaningless. Sufficient discussion on the activity selection process is necessary from various viewpoints; how they fit into the development environment, how they integrate with the management methods and training plans, etc. and how the help achieve the objectives, provide focus for the questions and affect the definition of the metrics.

For process control, a review checklist is prepared for each phase of development based on the metrics specified by the GQM model and past history, e.g. prior fault data. Feedback to the process should also be performed as soon as possible after a review. Problems can be easily found using the review checklist. Periodic checks; e.g. monthly, or at the final review of each development phase, are required to monitor the process. The earlier the phase at which monitoring starts, the more effective it is for quality improvement. Audit and configuration management are also process control methods governing quality.

(Step 4) Measure and assess the process and the products

Project data will be collected periodically, at least at the end of each development phase. Based on the metrics selected, the process and the products are measured. The results are assessed by using specific rating criteria. It is helpful for manager to take proper action quickly. Continuous measurement and assessment can produce high quality product.

For visual management, graphical displays of the appropriate management information can be selected based on the graph selection form. The project's current status can be found by using the visual display tool provided by the SMC system. It is helpful for software managers to see the achieved quality level in a concrete form to support such activities as decision making, the management of quality and scheduling of the next workload. For example, graphs provide the manager with time series data indicating process and product changes, as well as comparative data from past projects.

(Step 5) Support corrective action

For low scoring metrics, some action should be taken. A corrective action list is prepared and used to improve both the current and future process and products.

Based on the assessment of results at step 4, proper action is required quickly for problems or the sign of any problems. If necessary, the activity plan can be revised. These experiences are accumulated and used to future projects.

Guidelines necessary to perform project management, based on SMC, are as follows:

- Goal selection
- Question selection
- Metric selection
- Activity selection
- Management graph selection
- Project status diagnosis
- Corrective action recommendation
- Reliability prediction

The SMC helps the manager in the definition of an appropriate software engineering process during the GQM and activities selection phases (steps 2 and 3), by allowing the manager to tailor goals, measures, methods and tools to the specific system/project characteristics. A data base can be defined and built to support the measurement environment during the GQM selection phase and to support both the development and management environments during the activities selection phase. After executing one whole cycle through the SMC process, the results of analyzing the current project data can be fed back to each SMC phase. Updating the database and improving each step of the SMC helps generate a software engineering process for future projects.

The SMC support system is currently a prototype built on top of existing software packages. It consists of (1) a data base, (2) a set of statistical packages, and (3) a set of graphical types (developed using Microsoft Excel), all integrated under a common user interface.

Accumulation of application information in a data base enables the organization to establish guidelines for future projects. Therefore, the relation between the system characteristics and the measurements associated with the particular GQM should be collected and saved in a data base. Emphasis should be on the metrics common across several projects

## 5. An example G/Q/M

A simplified pair of GQM models, one for product and one for process are given. They are written from the point of view of the manager (which may include some of the concerns of the customer) for evaluating various components to improve quality, cost and usage of methods based upon managerial data.

First we will define some terms and offer a model of the qualities of interest:

DEFINITIONS:

Size (NCSS) = the number of non–commentary source statement (NCSS)

Actual Effort (AEF) = total number of staff hours to develop a component

Estimated Effort (EEF) = estimated number of staff hours based upon the software science metric, E

Actual Errors (AER) = the total number of errors reported

Estimated Errors (EER) = the estimated number of errors based upon the software science metric, B

Actual Error Rate (AERR) = AER / NCSS

Estimated Error Rate (EERR) = EER / NCSS

Changes (CH) = the total number of changes reported

Change Rate (CR) = CH / NCSS

Effort Distribution (PED) = the percent of staff hours for a particular component spent in each phase

Test Efficiency (PTE) = the percent of machine time spent testing a component

Work Rate (WR) = NCSS / AEF

Effort Variance (EFV) = AEF / EEF

Error Variance (ERV) = AER / EER

MODEL:

The objectives for management are cost, quality and the effectiveness of the methods. Evaluation is performed on the basis of improvement over some norm.

Cost can be assessed as the relationship between input, staff effort, and output, the quantity of documentation and program produced. In this case we will consider cost as demonstrated by two factors: work rate (WR), which provides some measure of the cost of production for a line of code, and Effort Variance (EFV), which provides some measure of whether the effort is reasonable relative to some measure of the expected effort.

Quality is assessed in two categories, must–be quality and attractive quality. These terms, must–be quality and attractive quality, are common Japanese quality perspectives. Must–be quality means the fundamental qualities necessary for software to function, i.e., functionality and reliability. Attractive quality means any additional quality characteristics for the software to satisfy the users specific needs, e.g., usability, security, portability. In this case, we will consider quality as demonstrated by two factors: error variance (ERV), which provides some measure of whether the error rate is reasonable relative to some measure of expected errors, and change rate (CR), which provides some measure of the entropy of the system.

Method characteristics are assessed based upon their adherence to a set of standards. Project manager experience is also assessed since the success of a project deeply depends on his ability. In this case, we consider method evaluation using two factors: effort distribution (PED), which will provides us some insight into whether the distribution of the effort was acceptable according to standard baselines of effort distribution, and test efficiency (PTE) which when combined with test time, will provide some insight into the effectiveness of the test process, and therefore the effectiveness of the methods used for development.

Note that the model uses the software science measures, E and B as a basis for estimating, effort and bugs. It assumes these calculated values as basic estimates for the variables effort and errors and uses them as norms when comparing the actual values for effort and errors.

In our proposed model, the values of these variables for any component are then compared to the values for some normal population. All values within 2 sigma variation from the average are considered acceptable. Those values with more than a 2 sigma variation in the "right" direction are considered good; those with more than a two sigma variation in the "wrong" direction are considered as not meeting the target goal. For example, the effort variance (EFV) for a component is considered bad if it is greater than two sigma above the norm determined by the average value of cost for the rest of the population.

In the example given in the next section the baselines are determined by the the rest of the component population in the particular project. In an environment where there is data from a sufficient number of projects, the baselines could be determined by projects with similar characteristics from other projects.

PRODUCT GOAL:

Purpose: Evaluate various software components within a project in order to assess them and recommend areas for improvement.

Perspective: Examine the relative cost and quality from the point of view of the manager.

PRODUCT DEFINITION:

Product Dimensions: A quantitative characterization of the physical attributes of the product.

Q1. What is the size of each component in terms of non–commented source statements (NCSS)?

Q2. What is the value of the software science metrics for each component (E,B)?

Changes/Defects: A quantitative characterization of the enhancements, errors, faults, and failures.

Q3. What is the number of defects associated with each component (AER)?

Q4. What is the number of changes associated with each component (CH)?

Q5. What is the fault rate, change rate (AERR, CR)?

Cost: A quantitative characterization of the resources expended.

Q6. What is the staff effort involved in the development of each component, i.e. design, code, test?

Q7. What is the distribution of effort spent in the design, code and test phase (PED)?

Context: A quantitative characterization of the customer community and their operational profiles.

[No questions for this example]

In general, five viewpoints are necessary for process questions. Two of five, "Effort of Use" and "Effect of Use", are actually used in a case study next section.

PROCESS GOAL:

Purpose: Evaluate the design, code and test processes in order to improve them.

Perspective: Examine the relative cost distribution and test efficiency from the point of view of the manager.

PROCESS QUESTIONS:

Quality of Use: A quantitative characterization of the process and an assessment of how well it is performed.

Q8. How much experience does the team have with respect to the methods and tools used?

Q9. How much experience does the manager have with respect to similar projects?

Domain of Use: A quantitative characterization of the object to which the process is applied and an analysis of the process performer's knowledge concerning this object.

Q10. How understandable are the requirements?

Effort of Use: A quantitative specification of the quality perspective of interest. In this case, a quantitative specification of the costs.

Q6. What is the staff effort involved in the development of each component, i.e. design, code, test?

Q7. What is the distribution of effort spent in the design, code and test phase (PED)?

Q11. What is the machine time spent in the test phase for each component (PTE)?

Feedback from Use: This includes questions related to improving the process relative to the quality perspective of interest.

Q12. What is the input to the design and code methods and tools, and the defect detection methods and tools?

Q13. What should be automated?

## 6. Case Study

The concepts of SMC can be applied to a variety of project types because of the flexibility of this methodology. Metrics and development methodologies are tailored to each project. In this section, we discuss several general issues in applying SMC and provide a sample application to the management of a specific project based upon models and the goals, questions and metrics of the previous section.

In executing SMC in a project, the software management procedure mentioned previously, the templates, guidelines and some forms are used. A step by step approach based on this procedure is demonstrated. A sufficient budget for managing these activities is required. It is also necessary to establish an organization to support the SQM process. Certainly, a seminar on SMC for both managers and developers would have provided better results. It should be remembered that the more experience the manager and the organization have with SMC, the better they will be able to apply the method. The continuous application of the method provides a better support for quality and productivity.

This example uses the NASA/SEL [McGa85, Bas85b] project data base. Thirteen newly developed components for a particular project were selected. Size range of non-comment source statements is from 60 to 299 LOC. Graphs for project management were made using Microsoft Excel.

In step 1, "System/Project Form" is filled out. This clarify both the software functional requirements and the development environment. The profile of the system and the environment are defined.

In step 2, the project goals are determined based on the system/project characteristics from step 1 and the managerial strategy; e.g. cost, quality level to be achieved, methodologies to be

employed, etc. Each goal is extended to questions and metrics by means of the GQM template. The "Quality Target" and "Managerial Metrics" are determined at this step. Cost and quality improvement and better usage of various methods were chosen as goals. To support management, the "Graph Selection Form" is provided. Six graphs were selected; those are for work rate, effort variance, error variance, change rate, effort distribution and test efficiency. Questions to achieve these goals are shown in Chap. 2.

In step 3, the best way to achieve GQM is discussed and appropriate activities are selected. These depend on the pieces of information from previous steps. Development methodologies and quality checkpoints are listed on a specific form. This form is used as a checklist during development.

In step 4, the development process is monitored and managerial data are collected periodically. To make the project status visible, display graphs are very helpful. The graphs used were selected in step 2.

The following table shows the results of statistical analysis on the NASA/SEL project. Six criteria on three categories are chosen. Regression analysis was executed for the "Error Variance" data. Analysis of variance was executed for the rest of data. Based on the graphs and this table, the project's current status can be found. Comments for four of 13 components are described below. Figure 2 shows some sample graphs.

Rules for interpreting the results

For each metric, there exists pattern to interpret the results. Consider the following examples.

[ Cost ]

(1) Work Rate: Development speed measured by NCSS per man–hour

- In case of a low value, there are several potential problems
  * low quality
  * insufficient development environment
  * loose process control

  etc.

(2) Effort Variance: actual effort vs. estimated effort

- Evaluate the goodness by variation between estimated and actual effort
  * in the case that the actual effort is lower, the work rate is high (or functions could be simple)
  * in the case that actual effort is high, the interpretation of the results are the same as for Work Rate.

[ Quality ]

(1) Error Variation: the number of actual errors compared with the estimated (a measure of complexity)

| Component number | COST | | QUALITY | | METHOD | | | |
|---|---|---|---|---|---|---|---|---|
| | Work Rate | Effort Variance | Error Variance | Change Rate | Effort Distribution D | C | T | Test Efficiency |
| c29 | | | | O | x | x | XX | XX |
| c61 | O | | | | | | | |
| c6 | | X | | | | x | XX | |
| c5 | X | XX | OO | | X | | x | O |
| c46 | | | XX | | | | | |
| c7 | | | XX | XX | XX | x | | OO |
| c9 | | | | | | | | XX |
| c4 | | | O | | | | | |
| c49 | | | | XX | | X | | |
| c43 | OO | O | OO | O | | X | | OO |
| c11 | | X | XX | | | X | | XX |
| c63 | | | OO | | | x | | XX |
| c50 | | | | | | | | O |

D design phase
C code phase
T test phase

Assessment Criteria (except Cost Distribution)

| | | |
|---|---|---|
| OO : | excellent | ( >= AVE +2 O~ ) |
| O : | good | ( >= +1 O~ ) |
| X : | poor | ( <= -1 O~ ) |
| XX : | bad | ( <= -2 O~ ) |

Assessment Criteria for Cost Distribution

| | | |
|---|---|---|
| XX : | very high rate | ( >= AVE +2 O~ ) |
| X : | high rate | ( >= +1 O~ ) |
| x : | low rate | ( <= -1 O~ ) |
| xx : | very low rate | ( <= -2 O~ ) |

**Table 2. Component Assessment Table.**

- It assumes that the greater the complexity, the greater the number of errors.
  * Quality is high if the number of errors is low in comparison with the estimated number based on complexity.

(2) Change Rate: the normalized magnitude of the number of specification changes and error modifications

- In case that the number of specification change is large, there is a problem in the development methods
  * insufficient review
  * less communication with user
  * loose configuration control
- In case that the number of error modification is large,

quality is considered to be low.
- In both cases, degradation of the system can be assumed due to entropy because of change

[ Methodology ]

(1) Effort distribution: effort ratio of each phase

- Evaluate the percentage of effort in each phase (design / coding / test).
  * Is the effort in the design phase sufficient?
    In the case of insufficient effort, the degree of specification completion is considered to be low.
  * Does it cost too much in coding phase?
    In case of too much effort, it is assumed that the specification is insufficient or the development environment is not so good.
  * Is the effort appropriate in test phase?
    In case of too little effort, it is assumed that testing was insufficient and the system was delivered with errors.
    In case of too much effort, it is assumed that the test method is not efficient and/or the quality is low so the test phase lasted too long.

(2) Test efficiency : percent of machine time in the test phase

- The ratio is high if the preparation of test cases is sufficient.
- The ratio is high if quality is high so error modification effort is small.

Assumed activities in the test phase

| Preparation | Machine Test ( fixed ) | Error modification |
|---|---|---|

The pattern for interpretating of results can be made by combining the above heuristics.

Comments

c5 :

Quality is good, but cost is high. Because of the high cost in design and code phases, product quality must be high. Some changes may have caused the rise of both design and code cost rate.

(good)
Quality is high.

(to be improved)
Work rate is low.

(diagnosis)
It is necessary to monitor the early process to avoid the slide of schedule. Quick feedback and effective

reviews are necessary.

c7 :
There is a problem in the methods for development and management.
The number of changes is large. This caused the rate of
design and code to be too high. Because of insufficient test
instead of high test efficiency, number of errors is also large.

(to be improved)
- Though test efficiency is very high, preparation,
interpretation and error correction must be insufficient,
because there are still many errors.
- There are many more changes than those of other components.

(diagnosis)
Design or review methodology must be improved. Be
more careful in test phase. Try to find out the potential
errors based on the test results. More experience and
knowledge are required to do so.

c43 :
It's a very good component. The only concern is the percentage
effort of the code phase. It is true that the difficulty
of this component is low, but both quality and productivity
are high.

c11 :
There is a problem in methods for development and management.

(to be improved)
Because of poor design, the code phase costs too much
and there are many errors.

(diagnosis)
It is necessary to improve design phase to be able to make
a better quality document. The test method should also be
reconsidered.

In total, the difference between the goals and results can be evaluated from Table 1.

From the view of COST, only component 5 was well above the standard cost. This component, however, achieved a high quality rating, so its project goal can be considered as achieved.

From the view of QUALITY, four of thirteen components (7,11,46,49) did not realize their quality target. Error analysis indicates that most errors can be reduced by avoiding careless mistakes. Component 7 has an extra problem. An unusually high number of changes extended the design phase and caused many errors. Further investigative action should be taken into the causes of those changes and the manager should be encouraged to minimize change. The quality target has not been achieved for these four components.

From the view of USAGE OF METHODS, two components (6, 29) had too high a cost in test phase. They rated satisfactory for cost and quality however. Four component (9, 11, 29, 63) rated poorly with respect to test efficiency. One component (29) did not meet target in both

categories. There are several problems to be solved in test phase.

All three goals can not always be achieved sufficiently. However, avoiding careless mistakes and improving the test method should produce a better product.

In step 5, corrective action is taken based on the collected data and managerial graphs from step 4. For unachieved items in Figs. 4 and 6, the cause of each problem is pursued and an improvement method is discussed and executed. If something is found wrong in a certain step, the activities in that step are improved quickly. In this way, a project can be managed systematically throughout the life cycle.

The expected effects of applying SMC are quality improvement, managerial cost reduction, accumulation and reuse of experience and a highly visible management reporting system.

## 7. Evaluation of Software Management Cycle

We are interested in evaluating and improving the SMC itself. Data collected at each phase and after release enable us to analyze the effect of the SMC. The followings are the GQM for evaluation of SMC.

Goal: Evaluate the effectiveness of the SMC

Process Conformance:

Q1. How much managerial training was given to the manager?
Q2. How well were the SMC methods applied?

Domain Conformance:

Q3. How well was the SMC procedure understood?
Q4. How well was how to interpret graphs understood?

Cost:

Q5. How many hours were spent to perform SMC?

Effect:

Q6. What was the distribution of the management time?
Q7. Were graphs and forms helpful for the manager?

Feedback:

Q8. What changes need to be made in the methodology to make it more effective?
Q9. What tools or activities would make the use of SMC more effective?

During development, quality/productivity metrics (set Q), methods metrics (set M) and

feedback metrics (set F) are necessary. Customer satisfaction metrics (set C) are required after release.

Table 2 shows the classification of data.

Four evaluation methods are provided.

(1) How good are goals?

Based on correlation analysis between Q, M, F and C, it can be judged that a project must be good if the metrics of the project include most of elements of Q, M or F which have high correlation coefficient to C.

(2) How good are activities (methods, feedback)?

Based on correlation analysis between M, F and C, it must be good activity if an element of M or F has high correlation coefficient.

(3) How good are metrics?

Based on regression analysis between C and Q, M, F, the metrics of a project must be good or predictable if the project has high regression coefficient.

(4) How good are products?

Based on significant test of the difference between two population (past projects' C and current C), the current projects' products must be good if the difference is statistically significant.

The results of these analyses help to improve Software Management Cycle and update the knowledge of management database.

## 8. Conclusion

The concepts and use of Software Management Cycle based on the Quality Improvement Paradigm are described in this paper. This methodology can improve not only product quality but also process quality. Three concepts; goal–oriented measurement, process control and visual management, are important to manage a project effectively, quantitatively and objectively.

Further plans for the SMC include:

(1) its application to a variety of projects, analyzing the processes and accumulating knowledge for different project classes, and

(2) the development of a full management support tool which covers the whole process.

The authors are convinced that this methodology contributes to the building of an appropriate software engineering process for improving both quality and productivity.

## 9. References


[AzSu86] M.Azuma, T.Sunazuka, "Software Quality Measurement and Assurance Technology (SQMAT)", Quality, Vol.16, No.1, pp.79–84, January 1986, (in Japanese).

[AzSM87] M.Azuma, T.Sunazuka, K.Minomura, "Software Quality Assessment Criteria and Measurement Technology", Standardization and Quality Control, Vol.40, No.8, pp.63–75, August 1987, (in Japanese).

[Bas85a] V.R.Basili, "Quantitative Evaluation of Software Engineering Methodology," Proc. of the First Pan Pacific Computer Conference, Melbourne, Australia, September 1985 [also available as Technical Report, TR–1519, Dept. of Computer Science, University of Maryland, College Park, July 1985].

[Bas85b] V.R.Basili, "Can We Measure Software Technology: Lessons Learned from 8 Years of Trying," Proceedings of the Tenth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, December 1985.

[BaKa83] V.R.Basili, E.E.Katz, "Metrics of Interest in an Ada Development," Proc. of the IEEE Computer Society Workshop on Software Engineering Technology Transfer, April 1983, pp. 22–29.

[BaRo87] V.R.Basili, H.D.Rombach, "Tailoring the Software Process to Project Goals and Environments," Proc. of the Ninth International Conference on Software Engineering, Monterey, CA, March 30 – April 2, 1987, pp. 345–357.

[BaRo88] V.R.Basili, H.D.Rombach, "The TAME Project: Towards Improvement–Oriented Software Environments," IEEE Transactions on Software Engineering, vol. SE–14, no. 6, June 1988, pp. 758–773.

[BaSe84] V.R.Basili, R.W.Selby,Jr., "Data Collection and Analysis in Software Research and Management," Proc. of the American Statistical Association and Biomeasure Society Joint Statistical Meetings, Philadelphia, PA, August 13–16, 1984.

[BaSe85] V.R.Basili, R.W.Selby,Jr., "Calculation and Use of an Environment's Characteristic Software Metric Set," Proceedings of the Eighth International Conference on Software Engineering, London, UK, August 1985.

[BaWe84] V.R.Basili, D.M.Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, vol. SE–10, no.6, November 1984, pp. 728–738.

[BaWe81] V.R.Basili, D.M.Weiss, "Evaluation of a Software Requirements Document by Analysis of Change Data," Proceedings of the Fifth International Conference on Software Engineering, San Diego, USA, March 1981, pp. 314–323.

[BoBL76] B.W.Boehm, J.R.Brown, and M.Lipow, "Quantitative Evaluation of Software Quality," Proceedings of the Second International Conference on Software Engineering, 1976, pp. 592–605.

[Grad87] R.B.Grady, "Measuring and Managing Software Maintenance", IEEE Software, Vol.4, No.5, pp.35–45, September 1987.

[GrCa87] R.B.Grady, D.L.Caswell, "SOFTWARE METRICS: Establishing A Company–wide

Program", Prentice–Hall, Englewood Cliffs, NJ, 1987.

[McRW77] J.A.McCall, P.K.Richards, G.F.Walters, "Factors in Software Quality", RADC TR–77–369, 1977.

[McGa85] F.E.McGarry, "Recent SEL Studies," Proceedings of the Tenth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, December 1985.

[Mizu82] Y.Mizuno, "Software Quality Improvement", Proc. 6th compsac 82, 1982.

[Muri80] G.E.Murine, "Applying Software Quality Metrics in the Requirement Analysis Phase of a Distributive System", Proc. Minnow Brook Conference, 1980.

[RoBa87] H.D.Rombach, V.R.Basili, "A Quantitative Assessment of Software Maintenance: An Industrial Case Study," Conference on Software Maintenance, Austin, Texas, September 1987, pp 134–144.

[SuAY85] T.Sunazuka, M.Azuma,N.Yamagishi, "Software Quality Assessment Technology", Proc. 8th International Conference on Software Engineering, London, UK, pp.142–148, August 1985.

[WeBa85] D.M.Weiss, V.R.Basili, "Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory," IEEE Transactions on Software Engineering, vol. SE–11, no. 2, February 1985, pp. 157–168.