A MEASUREMENT SYSTEM FOR LARGE, COMPLEX SOFTWARE PROGRAMS

Kyle Y. Rone, Kitty M. Olson, Nathan E. Davis
IBM Corporation
3700 Bay Area Blvd.
Houston, TX., 77058-1199

ABSTRACT

This paper describes measurement systems required to forecast, measure, and control activities for large, complex software development and support programs. Initial software cost and quality analysis provides the foundation for meaningful management decisions as a project evolves. In modeling the cost and quality of software systems, the relationship between the functionality, quality, cost, and schedule of the product must be considered. This explicit relationship is dictated by the criticality of the software being developed. This balance between cost and quality is a viable software engineering trade-off throughout the life cycle. Therefore, the ability to accurately estimate the cost and quality of software systems is essential to providing reliable software on time and within budget.

Software cost models relate the product error rate to the percent of the project labor that is required for independent verification and validation. The criticality of the software determines which cost model is used to estimate the labor required to develop the software. Software quality models yield an expected error discovery rate based on the software size, criticality, software development environment, and the level of competence of the project and developers with respect to the processes being employed.

A MEASUREMENT APPROACH

Thirty years of experience with programs for the National Aeronautics and Space Administration (NASA) has shown that the primary key to customer satisfaction is the capability to concurrently and consistently deliver compliant products, on time, within budget, and with an appropriate quality level. Figure 1 illustrates a measurement approach that leads to stabilization and control of the project and, ultimately, to improving the processes involved.

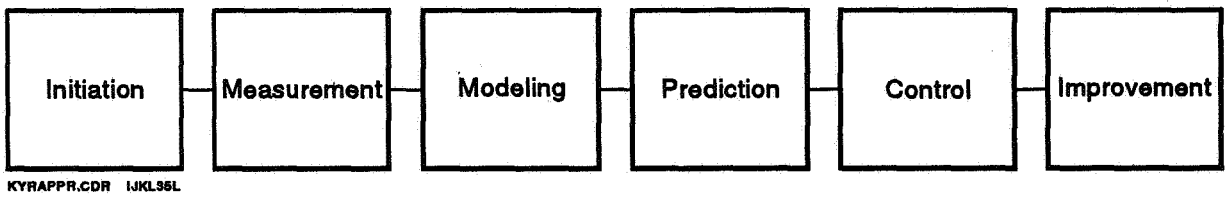

Figure 1. A measurement approach to stabilize and control a project.

Initiation consists of defining, tailoring, and stabilizing the required processes. Processes are institutionalized through standards, education, and procedures. Procedures based on process models which include precedence order and identified interim products permit visibility and control of the process.

Measurement includes defining a consistent set of measures which relate to the key goals of the project and the processes being utilized. It is important to measure with integrity. The purpose is not only to show that work is progressing, but to provide an understanding of how the processes are working. Measurements must be collected and stored in a manner to facilitate historical analysis.

Modeling involves presenting the data in a form that can be related to the process models and important, identifiable project parameters. These parameters include size, complexity, criticality, and process proficiency. The models must also be related to project schedules so that staffing needs can be analyzed.

Prediction begins by calibrating the new problem using historical data, tailoring the process to the new problem, and selecting an appropriate model. The model and the current product definition are used to forecast the key project parameters of size, complexity, and criticality. A plan is developed based on the process model and the predicted parameters.

Control points are established as key schedule milestones to permit assessments. After the previous steps are complete, the stage is set to activate the control loop: reaching a defined control point, collecting measurements, evaluating the data, acting on the data, and assuring that changes to the system are reflected in the plan.

Improvement can be initiated only after completion of these steps--initiation, measurement, modeling, prediction, and control. To improve the process, a change must be proposed, the impact on the key measurements predicted, and the first five steps repeated to re-establish and stabilize the process. Improvements can be assessed only when taken individually so that the impact of a change can be isolated. This approach applies to all the "keys to customer satisfaction" as illustrated in Figure 2. The consistent application of this approach is required to assure that all customer satisfiers are met.

Keys	Steps					
	Initiation	**Measurement**	**Modeling**	**Prediction**	**Control**	**Improvement**
Product	• Process • Interim product • Precedence order • Tailoring mechanism	• Size • Process proficiency	• Process models	• Process tailoring	• Control points	• Modify process • Modify ordering • Automation
Cost		• Function driven cost • Schedule driven cost • Complexity • Criticality	• Factor models • % Models • Phasing	• Calibrate to process • Function driven	• Cost management	• Modify cost models
Schedule		• Process elapsed time • Process order	• Schedule rules of thumb	• Phased cost and errors	• Schedule management	• Modify schedule rules of thumb
Quality		• Inspection errors • Process errors • Product errors • Total errors	• Life cycle errors • % Models • Phasing	• Calibrate to process • Function driven • Cost driven	• Quality management	• Modify quality models

Figure 2. Measurement approach applied to the keys to customer satisfaction.

COST AND QUALITY RELATIONSHIP

In developing software systems the cost and quality of a product can be traded against one another. By attempting to minimize development costs many projects simply defer error correction into the product time frame where the cost of error correction is more expensive. To prevent this from occurring a careful balance of product cost versus product quality must be established. The trade-off between cost and quality is dictated by the criticality of the function being developed. As criticality increases, it becomes imperative that the software be error free.

Reducing errors can be accomplished by thoroughly testing the software throughout the development life cycle. As shown in Figure 3 the cost models are imposed on a curve which relates the product error rate to the percent of the project labor that is required for independent verification and validation. Independent verification and validation involves monitoring the test strategy, plans, and procedures for a project and may also require an organization independent of the development organization to conduct system tests. The criticality level determines which cost model is used to estimate the labor months required to develop a software system. For example, a software component classified as low criticality will incur verification costs and indirect costs which are a relatively low percentage of the overall total development cost. In contrast, a software component which is classified as high criticality will incur verification costs and indirect costs which are a relatively high percentage of the overall total development cost.

Each cost model is associated with a specific product error rate. For example, the low criticality cost model is related to a product error rate of one error per one thousand source lines of code (KSLOC). In contrast, the high criticality cost model is related to a product error rate of one tenth (0.1) error per one KSLOC.

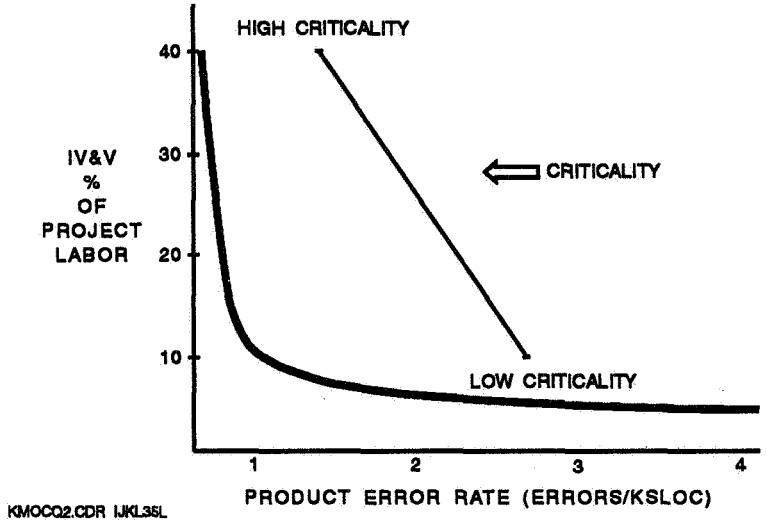

Figure 3. Product Error Rate versus Independent Verification & Validation Percentage (IV&V) of Project Labor

MODELING SOFTWARE COST

Modeling software cost is based on the technique of stepwise refinement. This technique involves decomposing the software system requirements into software functional components. These components are further decomposed into as many independent elements as possible. Decomposition terminates whenever a reused software element or a Commercial-Off-The-Shelf (COTS) software product is identified or whenever the component is decomposed to the lowest level. These software elements are sized and classified according to release, language, complexity, and criticality.

Software size, usually measured in source lines of code, is an important factor that ultimately affects the accuracy of the labor estimate. For example, as the size of the software system increases, a parallel increase in the interdependency among the various software components also occurs.

Release represents either an incremental product release, a release of the software development environment, or the learning curve associated with the software development process. Language is the programming language in which each software component will be implemented.

Complexity, the relative difficulty of developing each software component, is an important factor affecting development costs. Some types of software systems are more difficult to develop than others, e.g., developing an operating system versus developing utility software.

Criticality is the level of effect of a failure of a software component. Software for certain medical diagnostic or treatment systems and air traffic control systems must not fail or human lives could be lost. In contrast, an inventory control system should not fail, but the impact of the failure would not result in the loss of human life. As illustrated in Figure 4 these inputs--size, release, language, complexity, and criticality-- are used by the cost model to generate an estimate of the overall effort required to develop the software components and to determine how the effort will be distributed.

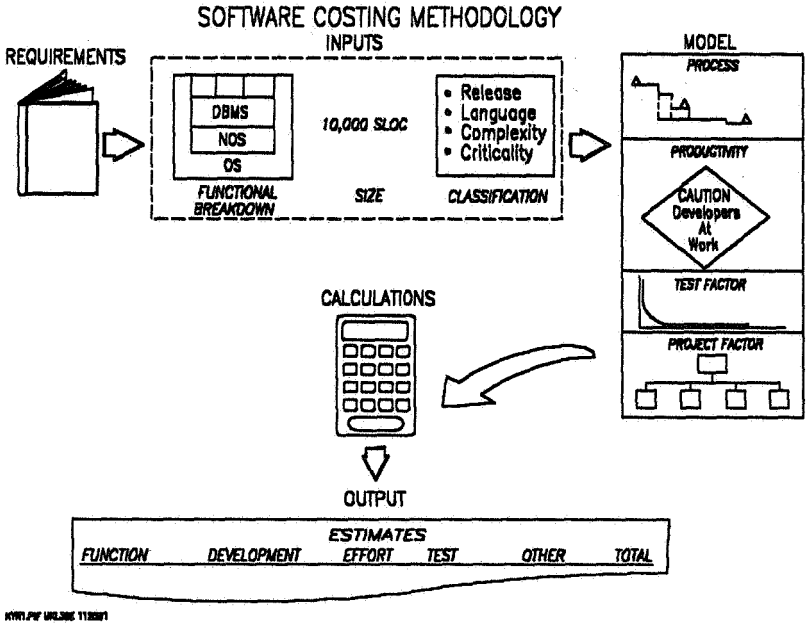

Figure 4. Software Cost Estimation Methodology

MODELING SOFTWARE QUALITY

The software development process is inherently complex; therefore, many opportunities exist to make errors. It is essential that an organization project an expected error distribution for each software development project. By plotting the actual error distribution against the expected error distribution curve, management can judge whether or not work is proceeding within expected bounds. Using this information, management can determine how well their error prediction and error prevention practices are working.

As in the cost estimation process, the requirements are first decomposed into functional components. These components are sized and classified according to release, criticality, project proficiency and development proficiency. Project proficiency represents the level of competence of a project with respect to its software engineering process. This is, how well a project as a whole is able to implement a software engineering process. Project proficiency determines how many errors will be inserted in the product per one thousand source lines of code. Development proficiency represents the level of competence of the developers with respect to their process. Development proficiency determines the total number of errors that will be discovered early in the development cycle. Development proficiency is dependent on situational factors such as experience level, availability of mentors, the ability to integrate new technology and especially on how well software product inspections are conducted. Criticality determines the project's product error rate, i.e., the number of product errors per one thousand source lines of code.

As illustrated in Figure 5, these inputs--size, release, criticality, project proficiency, and development proficiency--are used by the quality model to generate an expected error distribution pattern of early, process, and product errors.

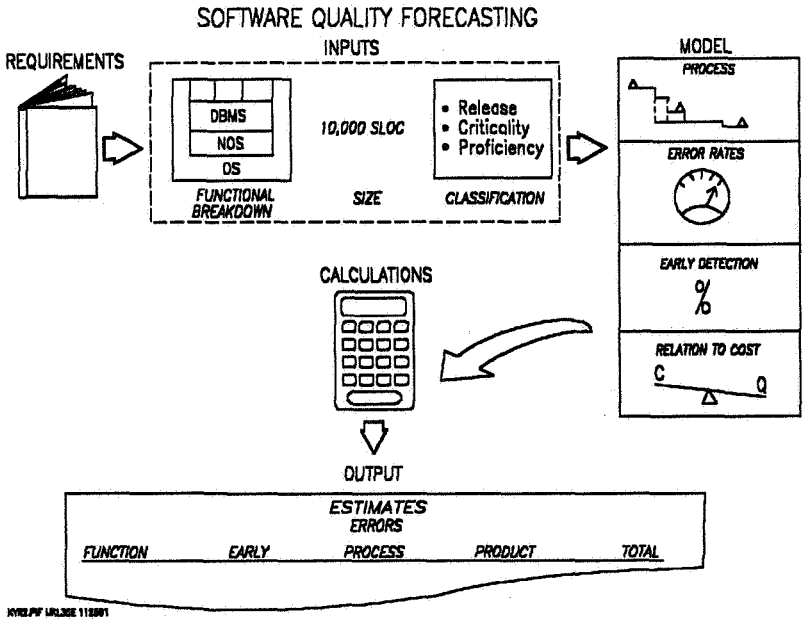

Figure 5. Software Quality Forecasting Methodology

MODELING A STAFFING PROFILE

Once the cost and quality estimates are determined a Rayleigh curve can be used to phase either estimate over time. A Rayleigh curve is a plot of a mathematical function which describes a life cycle phenomena. The Rayleigh curve illustrates whether the slope of a staffing curve is too steep or whether the error density is too great at certain points in the process.

Figure 6 illustrates the staffing profile for an ideal project. A minimum level of critical skills is required during the maintenance phase. This steady-state staffing level forms the support line. It includes critical skills for requirements, design, implementation, testing, and management. The support line is a function of system size and productivity as well as unique skill requirements specific to the software being maintained. In Figure 6 the area below the support line and above the maintenance tail of the Rayleigh curve represents the capability for new development work.

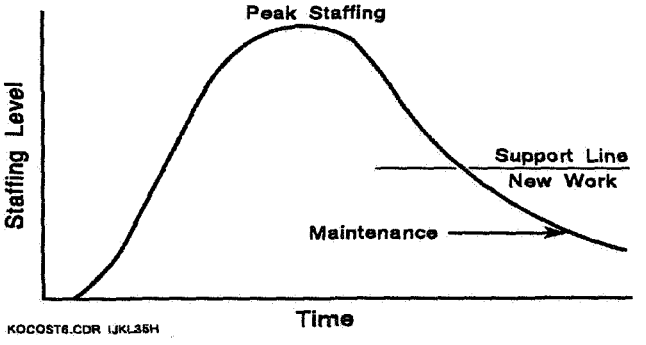

Figure 6. Staffing level is modeled using a Rayleigh curve.

As shown in Figure 7 the total maintenance effort can be modeled as the sum of a sequence of Rayleigh curves. The sizing and scheduling of new development activities should be planned to provide a stable level of effort as illustrated by the total development line. Software maintenance which handles Problem Reports can continue at a lower support level as illustrated by the total maintenance line. The total development line should not fall below the critical skills required by the project as determined by the initial staffing model.

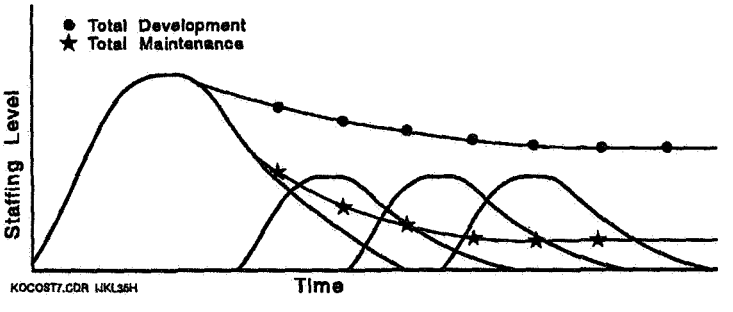

Figure 7. Maintenance effort modeled as a sequence of Rayleigh curves.

SUMMARY

Software now controls everything from the nation's telephone communication system to the world's financial systems. Delivering reliable software on time and within budget depends on an accurate measurement approach. An integrated measurement approach provides the information needed to effectively plan, manage, and control the software development process.

The cost and quality models described in this paper provide the capability to quickly generate estimates for diverse types of projects. Changes in assumptions such as size, complexity, or criticality are easily factored into the cost and quality estimates. The estimates can be phased across time to determine if the staffing profile or the error density is too steep at certain points in the process.

REFERENCES

Boehm, B. W., "Improving Software Productivity," *COMPUTER*, Vol. 20, No. 9, September, 1987, 43-57.

Kan, S. H., "Modeling and Software Development Quality," *IBM Systems Journal*, Vol. 30, No. 3, 1991, 351-362.

Kemerer, C. F., "An Empirical Validation of Software Cost Estimation Models," *COMMUNICATIONS of the ACM*, Vol. 30, No. 5, May, 1987, 416-429.

Putman, L. H. and W. Myers, *Measures For Excellence*, Yourdon Press, Englewood Cliffs, New Jersey, 1992.

Rone, K. Y., "Cost and Quality Planning for Large NASA Programs," *Proceedings of the Fifteenth Annual Software Engineering Workshop*, NASA Fifteenth Annual Software Engineering Workshop, Greenbelt, Maryland, Nov. 28-29, 1990.