15 S3-61 P-19

Automated Support for Experience-Based Software Management

N93-19164

Jon D. Valett

NASA/Goddard Space Flight Center
Software Engineering Branch Code 552
Greenbelt, MD 20771
internet jvalett@gsfcmail.nasa.gov
phone: (301) 286-6564
FAX: (301) 286-9183

Abstract

To effectively manage a software development project, the software manager must have access to key information concerning a project's status. This information includes not only data relating to the project of interest, but also, the experience of past development efforts within the environment. This paper describes the concepts and functionality of a software management tool designed to provide this information. This tool, called the Software Management Environment (SME), enables the software manager to compare an ongoing development effort with previous efforts and with models of the "typical" project within the environment, to predict future project status, to analyze a project's strengths and weaknesses, and to assess the project's quality. In order to provide these functions the tool utilizes a vast corporate memory that includes a data base of software metrics, a set of models and relationships that describe the software development environment, and a set of rules that capture other knowledge and experience of software managers within the environment. Integrating these major concepts into one software management tool, the SME is a model of the type of management tool needed for all software development organizations.

Keywords: software management, measurement, reuse of experience, management tools

1.0 Background

Good software management is generally viewed as a critical ingredient in successful software projects. One key aspect of good management is having access to the data that are necessary to understand the strengths and weaknesses of an ongoing development effort. To provide such access, a myriad of management-oriented tools have been developed. These tools typically allow the software manager to perform cost and size estimation, to plan a development project, to set up work-breakdown structures, and to provide other planning needs. Such tools are certainly useful, yet they do not provide the full scope of functionality required for a manager to effectively evaluate a software project.

Ideally, an experience-based software management tool would enable a manager to observe a project's progress, to compare that progress with other projects or with a model of how a project "normally" behaves, to predict key project parameters such as size, completion date, or errors, to assess the project's progress pointing out its strengths and weaknesses, and to analyze the quality of the software project and the software product. In order to provide this functionality, the tool would require access to key data relating to a project's status and to the past experience necessary to understand and manage the ongoing project. Included in this knowledge and experience is a data base of software metrics, a set of models of a development environment, a set of management rules that provide insight into a project's strengths and weaknesses, a set of quality definitions, and a set of relationships that help to define an environment's characteristics. Such a management tool would integrate this experience into a single environment providing the functionality required to actively monitor a software project.

A working model of the management tool described above is being developed within the Software Engineering Laboratory (SEL) at NASA's Goddard Space Flight Center (GSFC). This tool, called the Software Management Environment (SME) uses software measurement and the experience acquired from software measurement as its basis. Other tools either are being or have been developed that utilize measurement as a major component. These tools include TAME [1], Amadeus[2], and GINGER[3]. SME is a unique experience-based tool because it focuses on utilizing the measurement and the experience of a measurement program to automate support for project managers in actually monitoring the progress of their projects. While the SME has been constructed for a specific development environment, the concepts, architecture, and functionality of the tool, which are described in this paper, are general enough for any organization to build a similar tool. This paper will discuss the management activities that the SME addresses, the components needed to build an SME, and how these components are integrated to provide the management functions described.

2.0 Management Activities

In order for the SME to be an effective tool, it must automate key management functions. While the current SME is not comprehensive in its coverage of all management functions, it does provide support for many important aspects of software management. The SME utilizes a measurement-based approach to software management. Within this approach reusing management experience is viewed as an important aspect of the management process. This experience-based approach to management includes the following activities:

Observation and Comparison: The manager monitors the progress of a project by examining key project measures such as effort, size, and errors. The manager compares the status of the current project with past projects and with models of these measures that represent the nominal case within the environment. By observing and comparing, the manager is able to determine the current project's status and the differences between the current project and the normal project within the environment.

Prediction and Estimation: The manager estimates key project parameters such as project cost and size. The manager also, uses various models and relationships to continually update these predictions. These activities allow the manager to determine at-completion values for important measures and to estimate project schedule.

Analysis: Based on the measurement data, past project experience, and subjective information about a project, the manager identifies potential project problems.

Assessment: Using available measurement data and definitions of project quality, the manager assesses the overall quality of the ongoing project. For example, these quality assessments provide the manager with an idea of the project's maintainability, correctability, and stability.

A software tool should only attempt to automate aspects of a process that are understood well enough to perform manually; in the case of SME, all of the activities described above are carried out on projects within this development environment. In fact, such activities are part of the normal management process. The SME integrates data and experience into one tool that provides managers with functions that help them to perform these activities.

3.0 The Software Management Environment (SME)

The Software Engineering Laboratory (SEL) has actively been developing the management concepts that are the basis for the SME for the past 15 years. A prototype of the tool was developed between 1984 and 1987; this prototype provided a set of recommendations for developing an actual version of the tool.[4] This set of recommendations was then incorporated into the actual development of the SME, which began in 1987. The remainder of this section will discuss the SEL and the concepts that are the underlying ideas for the SME.

3.1 The Software Engineering Laboratory

The SEL was established in 1976 and has three primary organizational members: NASA/GSFC, Software Engineering Branch; The University of Maryland, Computer Science Department; and The Computer Sciences Corporation, Software Engineering Operation. The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effects of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices.[5] During the SEL's 15 years it has collected data on over 100 software development projects. These data include such items as software development effort, software size, error data, change data, and computer utilization data and are stored in a large repository called the SEL data base.[6] This data base has been used throughout the past 15 years to help the SEL to accomplish its three objectives. In the process of studying and measuring this particular development environment the SEL has produced numerous reports and papers which characterize this environment, evaluate various tools and methods, and capture experience and lessons learned in various software development efforts. (For a complete list of SEL documents and reports see the "Annotated Bibliography of Software Engineering Literature".[7])

Throughout the SEL's history, this software measurement program has been used extensively in the management of actual software projects. Such use of measurement data is common among companies that have instituted measurement programs (eg. reference [8]). As this use of measurement as a management tool evolved, the SEL began attempts to automate the process. Such automation is only possible through a comprehensive understanding of how to use software measurement data within a particular development environment. Within the SEL environment, software managers use not only the data collected on their current project, but also, the information and experience from past projects. The studies and reports characterizing the environment provide the manager with profiles of how particular measures behave, numerous relationships for estimation and prediction of such measures, and lessons learned concerning how to analyze measurement data. Automating the access to this vast corporate resource is the goal of the SME.

3.2 SME Concepts

Understanding the SME requires a firm understanding of the three major components that are the basis for the tool. The first is the SEL data base, it provides the historical data of past projects, as well as the dynamic data on projects that are currently being managed. The second, is a set of models and relationships that describe the development environment. These models and relationships provide the profile of a normal project, as well as the necessary information to predict and estimate key project parameters. Finally, experienced software managers analyze measurement data to determine a project's strengths and weaknesses. The knowledge required to perform this analysis is captured in management rules that provide the expert analysis portion of the SME. These three SME concepts provide the experience base needed for an organization to construct an SME-like tool.

An important aspect of these SME concepts is that the experience they represent continually evolves as the development environment and process changes. The SME packages the current level of experience; as it changes, the experience base is refined to reflect these changes. The representation of the experience, however, does not change. Therefore, the key aspect of the SME, from the perspective of someone who wishes to build a similar tool, is the concepts and the architecture of those concepts, not the experience itself.

Software Measurement Data

Measurement of the software development process and its products is a necessary component of successful software management. Within the SME, data from the SEL data base is utilized to provide the underlying measurement data. The SEL data base captures information on all software projects within one particular development environment. This data includes such items as the weekly effort expended on a project, the size of the ongoing software project (in both lines of code and number of modules), the amount of computer utilization on a project, and the number of errors uncovered as well as the number of changes made to the source code. In addition to these basic measures, the SEL data base contains data on such items as number of modules designed, number of open problem reports, and the amount of time spent uncovering and repairing errors. While these lists of data are not complete, they do provide a snapshot of the types of data available to the SME.

The SME uses the data from the SEL data base as a basis for all of its analysis, comparison, prediction and assessment. The data provide the information that characterize and describe the current software development project as well as past projects of interest. Having access to so much descriptive data allows the SME to provide its wide range of functionality. Thus, software measurement is the backbone of the SME. Measurement provides the basis for all other SME concepts; neither the management rules nor the models and relationships would be possible without it.

Models and Relationships

The second component of the SME is the models and relationships that represent the software development process and its products. The models and relationships used within the SME and presented within this paper are derived from numerous previous SEL reports and studies. A summary of the types of models and relationships used can be found in the document "The Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules".[9]

The term model is used to describe a pattern of how some measure or combination of measures normally behaves within a software development environment. Measurement models have been described in numerous SEL reports and papers, but they have generally all been developed using similar methods. Typically, a model for some particular measure is developed by examining the data for that measure over a set of similar projects. The data is then combined, usually using some type of averaging, to develop a model of the "normal" project. Since even within one environment all projects may not be homogeneous, different models for the same measure are developed for significantly different project types. Within the SME, there are currently two different model types, depending on the development methodology used on the projects. Other models may need to be developed depending on such parameters as project type, programming language, or development environment. Deciding what different factors constitute a distinct model type is an important research component of developing an SME. Certainly, each individual project is distinct, but usually projects within a development environment have many similarities that result in reasonable models.

As an example of a model that is used by SME, Figure 1 shows how source code grows within the SEL environment. (For the purposes of this paper, there is no need to distinguish between various model types.) It provides a representation of the typical growth of the number of source lines of code within a project's controlled library. The wide band indicates a range of what is considered to be "normal" source code growth. (In this case the range is one standard deviation on either side of the actual model.) As another example, figure 2 is the model of error rate for the SEL environment. This model shows the typical errors uncovered and repaired per line of code within the environment throughout a project's lifetime. Again, the band represents a range over which the error rate is considered "normal." (In both Figures 1 and 2, lines of code is defined as physical lines including commentary and blank lines. In Figure 2, error is defined as a conceptual error in the software.) Another kind of model used within the SME is of the amount of time spent in each phase of a project. This model is depicted in Figure 3; it provides a mechanism for determining how much calendar time a project normally spends in each phase of the software development life cycle.

Relationships, on the other hand, provide the SME with a way to estimate critical project factors based on other estimates, or current status. Relationships are typically developed by using numerous software development projects' data to determine if any correlation exists between various measures. Normally, such data analysis is done to test hypotheses that certain relationships exist between such measures.

As an example, within the SEL environment, a relationship has been found between lines of code and the actual duration of a project. This relationship is shown as the equation:

$$D = 5.450 * L ** 0.203$$

where,

D is the duration of the project in months (from project start through acceptance test), and

L is the total delivered lines of code in thousands.

Such a relationship allows a manager to estimate the length of a project based on an estimate of the number of lines of code for that project. Other relationships have been established between computer use and lines of code, effort and number of modules, etc. Such relationships provide a software manager both a mechanism for estimating various parameters and a consistency check for sets of estimates.

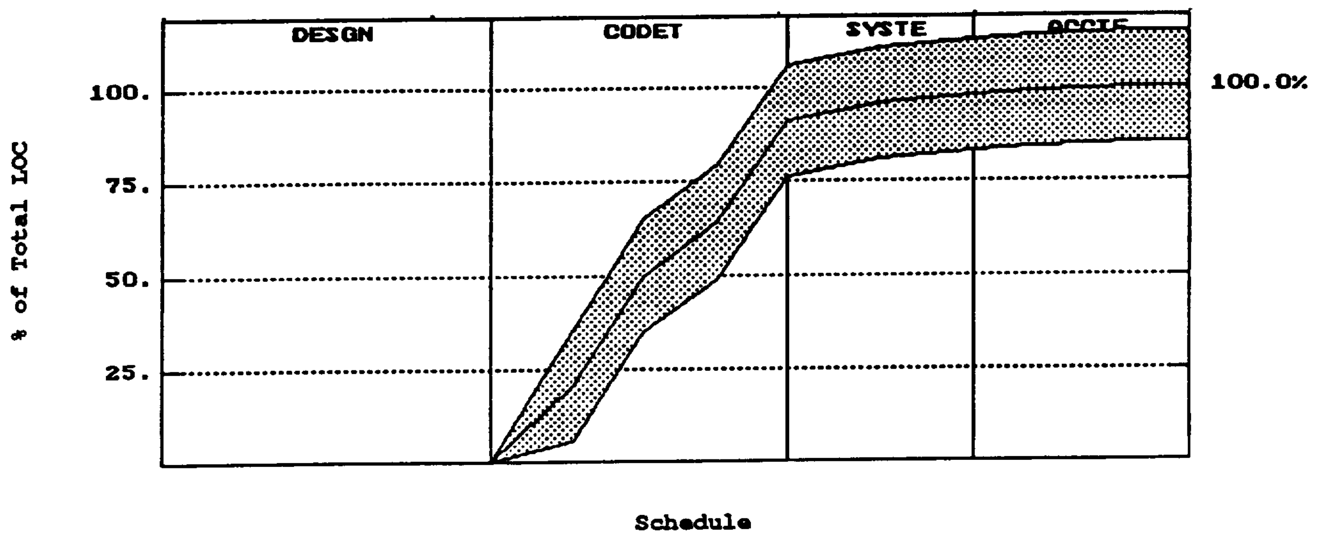

Figure 1: Model of Source Code Growth

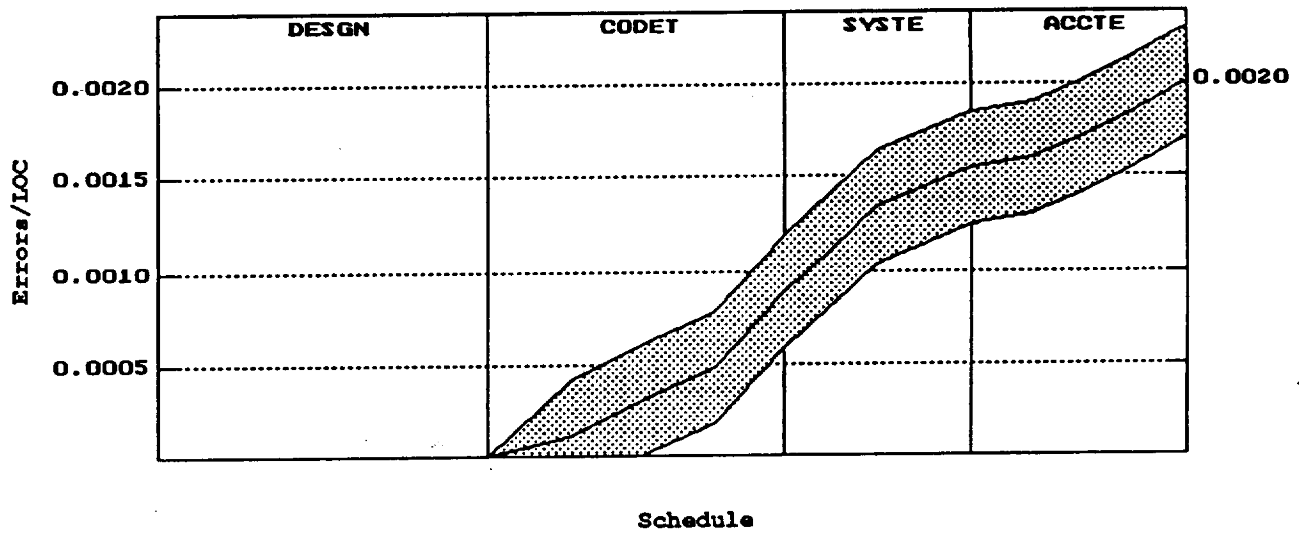

Figure 2: Model of Errors/Line of Code

Management Rules

Capturing how experienced software managers use and evaluate measurement data has been investigated by the SEL.[10] These studies show that using expert systems techniques for the capture and use of this experience is feasible in this domain. This knowledge about software measurement has been published in numerous SEL reports and it provides a foundation for creating an experience base for utilizing software measures in management.[9] The concept of these software management rules is that interviewing software managers and capturing how they interpret certain conditions of a project provides reusable knowledge concerning the strengths and weaknesses of a project. These interpretations are then combined into specific management rules that describe the possible explanations for certain conditions. For example, figure 4 shows a graphic of a simple management rule. This figure shows how one might interpret a deviation from the normal pattern of computer use per line of source code (again represented as a model similar to those described in the previous section). For example, early in the project if the number of CPU hours per line of code is above normal one possible interpretation is that the design was not actually complete. Later in a project, if the measure is below normal, the possible explanations might be either low productivity, or insufficient testing. Such a figure provides a simple representation of a management rule.

Actually, a number of simple management rules can be combined to form rules that describe the possibilities that certain explanations are true. For example, a rule such as

If the number of programmer hours per software change is above normal and

the project is early in the code phase then possible explanations are

Good solid, reliable code (0.5)

Poor testing (0.25)

Changes are hard to isolate (0.25)

Changes are difficult to make (0.25).

describes the possible explanations for a certain condition. This rule uses numbers to show the

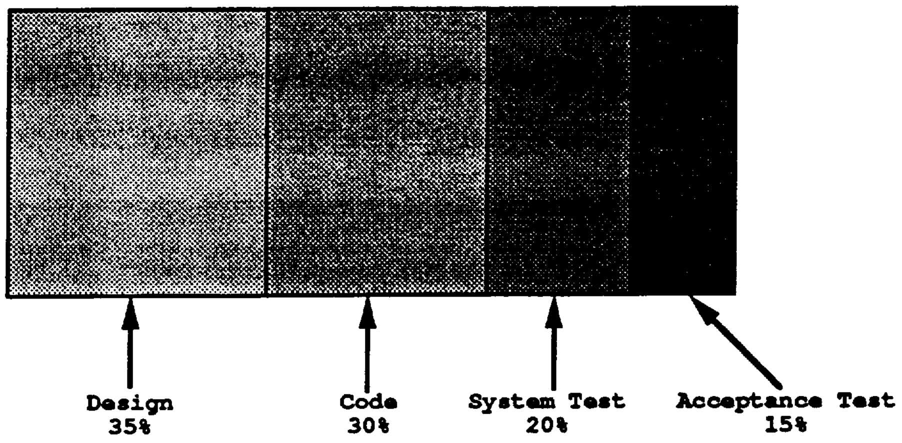

Figure 3: Model of Project Schedule

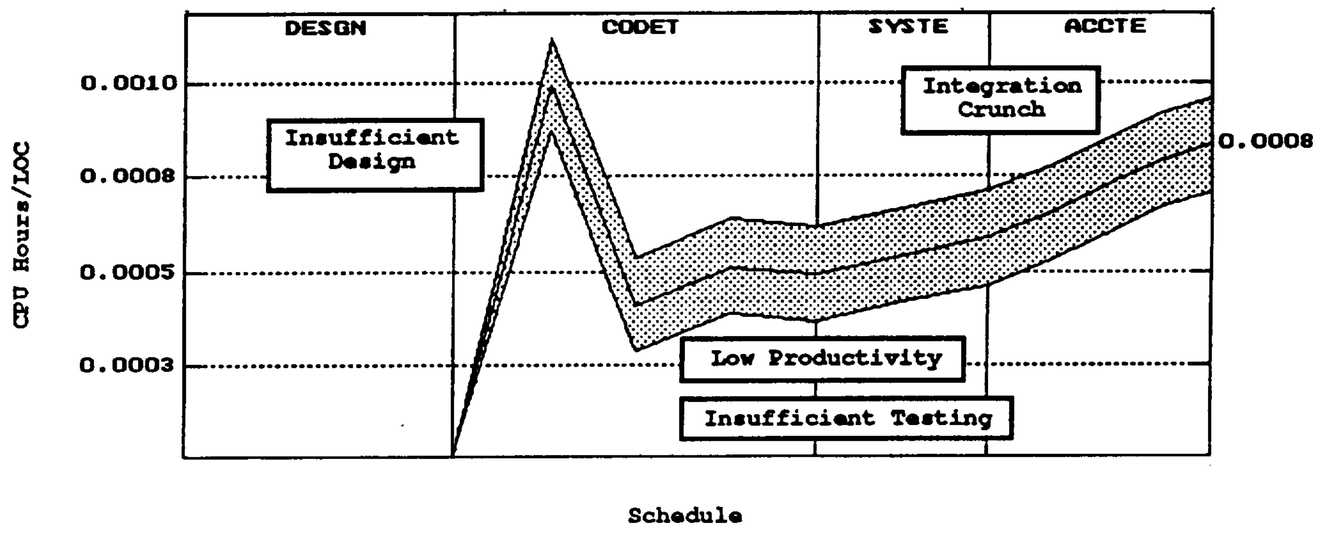

Figure 4: Rule for Analyzing Computer Use

certainty that each of the possible consequents are true. Thus, it is more likely that good solid, reliable code is the explanation for the deviation then poor testing, although either explanation could be true. This rule is then combined with other rules for other measure deviations to increase the certainty that particular explanations are correct. Using this method of evaluating software measures provides a set of possible explanations describing a project's strengths and weaknesses. By using sets of rules in this manner, an automated system can examine the empirical evidence about a project and provide some insight into the project's status.

4.0 Using the SME

This section describes how the SME utilizes the concepts described above to provide its functionality. While the concepts of the SME are the most important aspect of the tool, understanding how to utilize those concepts to provide management support is also of interest. Attempting to build an SME-like tool requires knowledge of how to integrate the experience into a useful tool. The examples used are realistic in that they show the actual functionality of the SME, however, due to the inability to reproduce the color SME images, the graphics images are in black and white.

Comparison

One major function of the SME is the ability to observe data and compare it to models and previous development efforts. Figure 5, shows an example of using the SME to compare data to a model. In this example the manager is looking at the way error rate behaves on the project of interest. The current project is shown as the solid line and the model is shown as a band of what is considered "normal" for error rate. The x-axis shows the expected schedule for the project. That is, the start date and end date shown are the manager's estimates, however, the other phase dates shown are the expected phase dates for the project (as calculated by the SME). The tool

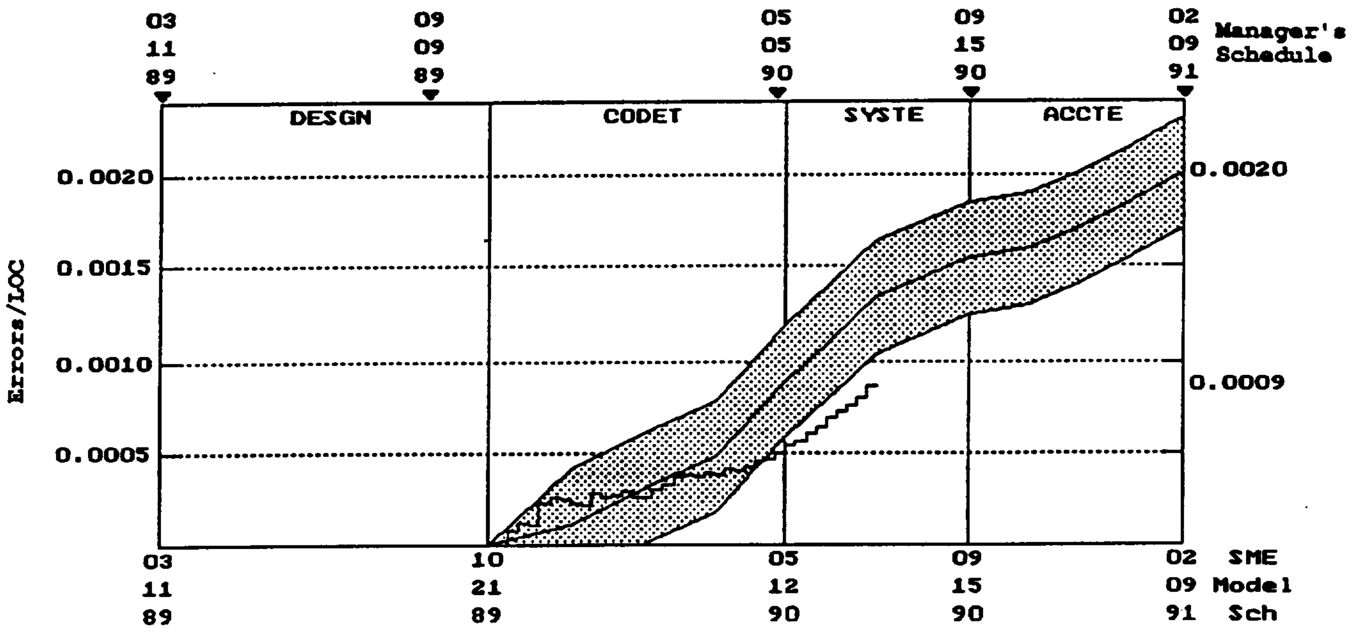

Figure 5: Rate of 'Reported Errors/Lines of Code' for Project A

also shows the manager's estimates for all the phase dates on the top of the screen. The Y-axis shows the error rate in errors per line of source code in the controlled library. Note that the phases represent a typical waterfall life cycle, with the major phases being design, code and unit test, system test, and acceptance test. By using this comparison, the manager is able to track such key items as error rate, productivity, and amount of computer time used. Additionally, the manager is able to overlay other projects' error rate patterns in order to compare the behavior of those projects to the current project.

<u>Prediction</u>

Figure 6, provides a look at another function of the SME. This figure is similar to the comparison figure, except that it also shows a predicted final value for the measure. In this figure, the measure of interest is computer use (in number of CPU hours). This is shown in absolute terms on the Y-axis. That is, the actual amount of time used on the machine is shown (it is not normalized). The SME allows the user to predict where the project will be when it is completed. This function utilizes the model and a projection of the progress of the project based on the measures in SME (eg. the project is 50% of the way through the code and test phase), to predict the final values of the measure, and of the schedule. In this example, the number of CPU hours on the project is predicted to be 1255, while the current estimate is 990 hours. Also, the project is predicted to take longer then the manager has estimated. Such predictions enable the software manager to gain another perspective on the final values of project measures and on the projected end date of the project.

<u>Analysis</u>

A key component of the SME is the utilization of expert systems technology for software management. Through experience, software managers are able to improve their ability to analyze software measurement data. Based on the measurement data and their experience, managers are able to identify the strengths and the weaknesses of a project. The SME utilizes a rule base that captures managers' knowledge of how to perform such analysis. This rule base is then used to analyze deviations from the normal project. An example of such analysis is found in figure 7. In this figure, the error rate of the current project is lower then normal for this particular point in the development life cycle. The SME uses this information, information about other measures, and subjective data about the project to provide possible reasons for such a deviation. The top two explanations are then displayed for the user. In this case, the explanations are that insufficient testing is being performed and that an experienced development team is producing a superior project. Either of these two explanations might be correct, they only provide insight to the user as to possible explanations for the deviations. Other explanations are certainly possible; the user of the tool can obtain further data on why the system reached its conclusions and on the other conclusions. The user can also provide the system with more subjective information about the project of interest, perhaps leading to changes in the conclusions that are inferred.

Assessment

A final function of the SME is to utilize software measures to provide an assessment of the overall quality of a software project. An example of such an assessment is shown in figure 8. In this figure the bar graph shows the SME's rating of certain quality measures as they compare to the normal project in the environment at that point in its development. The quality factors shown are maintainability, reliability, and stability. Each of these factors can be determined by combining various software measurement data. For example, the quality factor of maintainability is calculated by adding the percentage of errors that are easy to isolate with the percentage of errors that are easy to correct. Thus, as these percentages increase the maintainability of the project is said to increase. For each quality factor displayed, SME has a specific definition for how to compute that factor. These definitions, which are really a form of a relationship, use a specific set of measures to compute the relative value of that quality indicator. Of course, SME also uses a model of how these factors behave over time in order to display the normal band on the graph. Quality assessment provides the software manager with an overall appraisal of how the project of

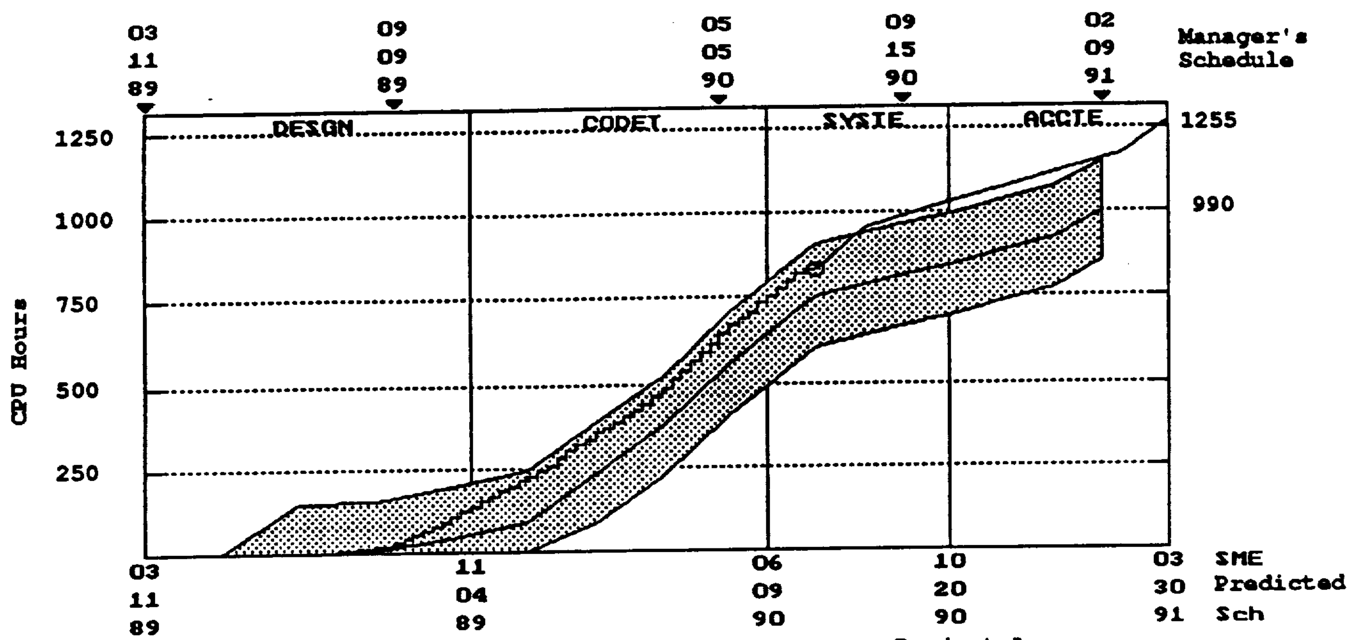

Figure 6: Predicted Growth in 'CPU Hours' for Project A

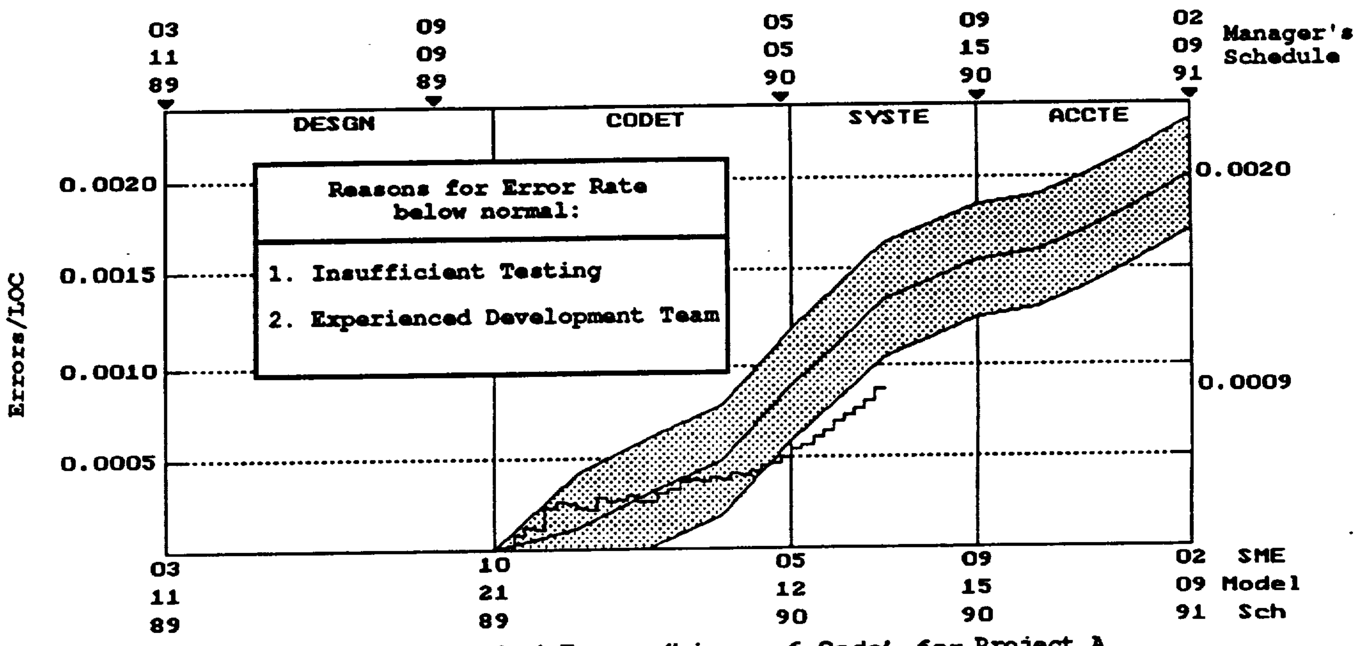

Figure 7: Analysis of 'Reported Errors/Lines of Code' for Project A

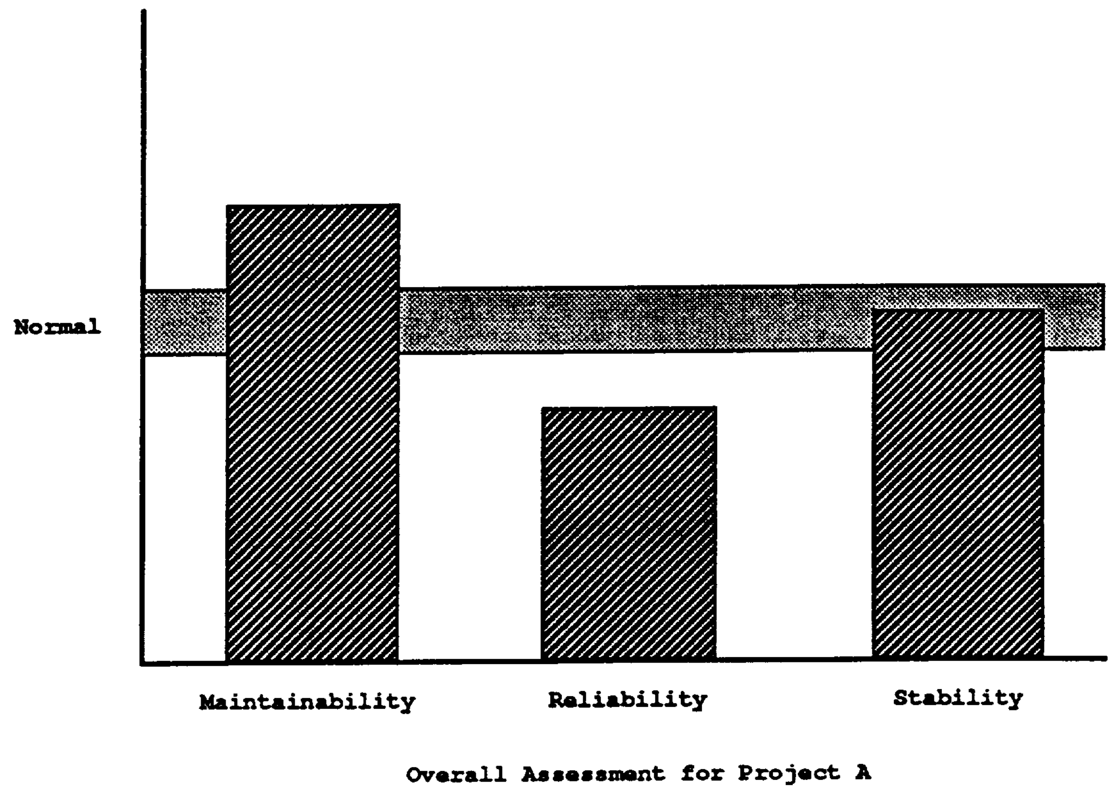

Figure 8: Overall Assessment Function

interest is doing compared to the normal quality measures in the environment.

5.0 SME as a Model Tool

Currently, the SME is being used by numerous software managers in the SEL software development environment to assist them in monitoring actual software projects. The SEL, as an experience factory [5], has provided the concepts necessary to build an SME for this particular software development domain. Other organizations can develop an SME-like tool by beginning to capture the experience of their environment. While within the SEL environment all three of the major components of SME have been well developed, other organizations may have only limited parts of the components. Such limitations should not be viewed as detrimental to the development of an SME. Similar tools should be developed using the experience available; they can then evolve into more complete tools as the local experience base provides additional artifacts for reuse.

The SME is an attempt to integrate a measurement process, the results of a longstanding software engineering research effort, and the expertise of software managers into a tool for managing and controlling software projects. As such, it provides for the utilization of corporate experience to manage ongoing software projects. An SME has been built for one particular software development organization. Other software development organizations should use the SME's concepts as a model for building similar tools for their environment. By providing the user with increased project awareness, predictions of key project parameters, expert analysis of software measures, and assessment of the overall quality of the development effort, an SME is extremely valuable to a software manager. Such a tool provides improved project management through the packaging of experience.

References

[1] Basili, V. R. and H. D. Rombach, "The TAME Project: Toward Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, June 1988, pp. 758-773.

[2] Selby, R. W., et al., "Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development," Proceedings of the 13th International Conference on Software Engineering, IEEE Computer Society Press, May 1991, pp. 288-298.

[3] Kusumoto, S., et al., "GINGER: Data Collection and Analysis System," Technical Report, Osaka University, Osaka, Japan, June 1990,

[4] Valett, J., "The Dynamic Management Information Tool (Dynamite): Analysis of Prototype, Requirements, and Operational Scenarios," Master's Thesis, University of Maryland, May 1987.

[5] Basili, V.R., et al. "The Software Engineering Laboratory - An Operational Software Experience Factory," Proceedings of the 14th International Conference on Software Engineering, IEEE Computer Society Press, May 1992.

[6] So, M. et al., "SEL Data Base Organization and User's Guide (Revision 1)," SEL-89-101, The Software Engineering Laboratory, NASA Goddard Space Flight Center, Greenbelt, Maryland, February 1990.

[7] Morusiewicz, L. and J. Valett, "Annotated Bibliography of Software Engineering Laboratory Literature," SEL-82-1006, The Software Engineering Laboratory, NASA Goddard Space Flight Center, Greenbelt, Maryland, November 1991.

[8] Grady, R., "Work Product Analysis: The Philosopher's Stone of Software?," *IEEE Software*, March 1990, pp. 26-34.

[9] Decker, W., R. Hendrick, and J. Valett, "The Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules," SEL-91-001, The Software Engineering Laboratory, NASA Goddard Space Flight Center, Greenbelt, Maryland, February 1991.

[10] Ramsey, C. and V. R. Basili, "An Evaluation of Expert Systems for Software Engineering Management," *IEEE Transactions on Software Engineering*, June 1989, pp. 747-759.