/2603
p. 9

An Information Model for Use in Software Management Estimation and Prediction

Ningda R. Li and Marvin V. Zelkowitz
Department of Computer Science
University of Maryland
College Park, MD 20742

Abstract

This paper describes the use of cluster analysis for determining the information model within collected software engineering development data at the NASA/GSFC Software Engineering Laboratory. We describe the Software Management Environment tool that allows managers to predict development attributes during early phases of a software project and the modifications we propose to allow it to develop dynamic models for better prediction of these attributes.

Keywords: Cluster analysis; Data modeling; Measurement; Software management; Tools

1 Introduction

Software management depends upon managers to collect accurate data of the software development process and on the production of accurate models upon which to use that data. Lines of code is still the most widely used measure for cost and error analysis, even though it is known to be inaccurate [8]. However, since it is not known until the completion of a project, its use as a predictive measure is not reliable. What are needed are more accurate models of the software development process.

Current models are developed according to broad categories, such as waterfall development, spiral model development, cleanroom development, etc., with additional qualifiers giving a few attributes of the product (e.g., real time, embedded application, data base). Data is often collected and projects are compared to historical baselines according to these general categories. For example, the COCOMO model [1] is based upon a small set of predefined factors, and predictions are made according to how a new project measures up to these factors.

Second International Conference on
Information and Knowledge Management
Arlington, VA
November, 1993

It is difficult for software managers, however experienced they are, to evaluate the status or quality of a software development project and make correct decisions without accurate, reliable measurement models and data. These data include metrics aimed at clarifying and quantifying some quality of either a software product, or the development process itself [13].

Since we do not have accurate models of the software development process, perhaps, we can use the data itself to develop dynamic models of software development that reflect the changing nature of the development process. In this paper we study one particular modeling technique, cluster analysis, as a means for determining the underlying information model present in the collected software engineering development data.

The importance of software management has led to the development of various software management tools for aiding in this effort. These tools help software managers get access to, visualize, and analyze measurement data. The Software Management Environment (SME) is one of those tools developed within the NASA Goddard Space Flight Center Software Engineering Laboratory (SEL) [6], [12]. It is the purpose of this paper to investigate the use of cluster analysis within SME to enhance the ability of software managers to predict and control the software development process.

In Section 2 we describe the information model and the measures used by SME. In Section 3 we describe our use of cluster analysis to dynamically change our information model, and in Section 4 we describe some preliminary results of using our new model. We then give our conclusions to this work.

2 Measurement in SME

For over 15 years the software engineering community has been studying various models of the software development process. Concepts like Halstead's software science measures, Putnam's Rayleigh curve, Boehm's COCOMO model, among many others, are all attempts at providing a quantitative model underlying the software development cycle. Unfortunately, most of these models are very general, and while broadly describing the software process, do not have the granularity to make accurate predictions on a single software project.

As a way to further these studies, the Software Engineering Laboratory was established to evaluate the above models and develop new models within a production programming environment.

2.1 NASA/GSFC SEL

The NASA Goddard Space Flight Center Software Engineering Laboratory is a joint research project of GSFC Flight Dynamics Division, Computer Sciences Corporation and the University of Maryland. Data from over 100 projects has been collected since 1976 and a data base of over 50 Mbytes of measurement data has been developed. Initially supporting 100,000 line FORTRAN ground support software for unmanned spacecraft written by 10 to 15 programmers over a 2 year period for an IBM mainframe, the SEL data base now includes a wider variety of projects consisting also of Ada and C code for a variety of machines.

The SEL collects data both manually and automatically. Manual data includes effort data (e.g., time spent by programmers on a variety of tasks - design, coding, testing), error data (e.g., errors or changes, and the effort to find, design and make those changes), and subjective and objective facts about projects (e.g., start and end completion dates, goals and attributes of project and whether they were met). Automatically collected data includes computer use, program static analysis, and source line and module counts.

2.2 Measure Models

Data modeling often combines various measures in order to evaluate attributes in a software development. For example, classification trees were used as part of the Amadeus project [9][10] and a variant of that method was used within the SEL [11]. In this case, a tree is generated where each leaf node represents one of several results. Based upon values for each measure, a path down the tree is taken until a result at a leaf node is reached.

For each project, we can compare the collected data over time with a predefined model of a similar project from the data base. A *basic measure model* refers to the expected behavior of a software development measure as a function of time [5]. Measures, developed from the raw data collected by the SEL, include lines of code, staff hours, computer hours, and changes and errors. A measure model is usually obtained by examining the data for that measure over a set of projects and averaging them. Time is described in terms of the four major phases of software development within the waterfall life-cycle: design, code and unit test, system test, and acceptance test.[1] Measure behavior is described in terms of percent completion of that measure at each distinct checkpoint.

Within the SEL, we describe one of these measure models as a vector of 15 points, each representing the percent completion of the measure at distinct dates in the development cycle (generally 25% increments through each phase). Table 1 shows the tabular representation of a Lines of Code (LOC) model [5] and Figure 1 shows the graphical representation of the same model. According to the LOC model, no code should be written during the design phase, and most of the code (76%) should be written during the code and unit test phase.

For ease of use, we can use the vector representation of the model:

$$[0,0,0,0,0,6.86,36.05,53.99,76.28,\\ 86.82,94.88,96.09,98.14,99.58,100]$$

In general, a measure model can be represented by the following vector:

$$P = [p_0, p_1, p_2, \ldots, p_{13}, p_{14}]$$

[1]The SEL does not collect specification data since that task is performed by another group. This is reflected in the models that the SEL develops, and is a good indication why no two development models are easily transportable across locations.

Phase	% of Phase	% of Total Lines
Design	0	0.00
	25	0.00
	50	0.00
	75	0.00
Code/Unit	0	0.00
Test	25	6.86
	50	36.05
	75	53.99
System Test	0	76.28
	50	86.82
Acceptance	0	94.88
Test	25	96.09
	50	98.14
	75	99.58
End	100	100.00

Table 1: Tabular representation of a LOC model

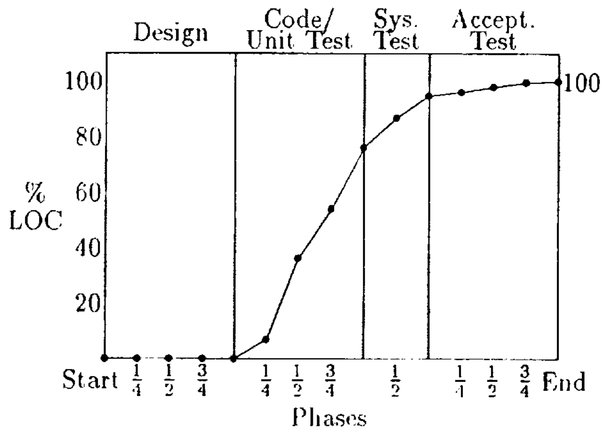

Figure 1: Graphical representation of a LOC model

with $p_0 = 0$, $0 \leq p_i \leq 100$ for $1 \leq i \leq 13$, and $p_{14} = 100$.

We will use ***measure pattern*** to refer to a measure model derived from a single project. Essentially, we produce a measure model as the average of some set of measure patterns.

2.3 SME

The Software Management Environment was developed to help software managers carry out management activities like observation, comparison, prediction, analysis, and assessment [6]. In order to provide these functions, SME uses measurement data from current and past projects from the SEL database, re-

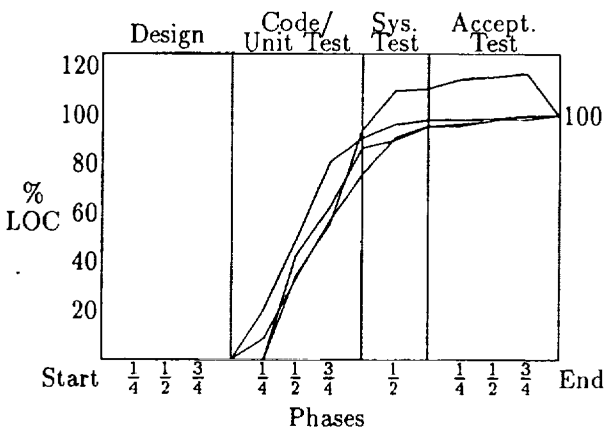

Figure 2: LOC patterns

search results in terms of models and relationships, and manager experience from the past.

SME was initially built with a fixed set of measure models. For example, for LOC (lines of code), the most apparent predictor seemed to be programming language. Therefore, SME originally had two models of LOC based upon language - Ada and FORTRAN. Each project was classified according to the measure model it was expected to adhere to, and for each measure type, a predefined measure model was stored in the data base.

Some of the features of SME are described below.

Measure models in SME

Currently in SME, a measure model is derived from a set of projects with the same characteristics, such as development methodology, programming language, and development environment. SME decides which measure model to use for a project measure of interest based on the characteristics of that project. For example, Figure 2 shows four LOC patterns of four different projects with the same characteristics. SME creates a LOC model by averaging these patterns, but is the resulting model a good representative of actual LOC behavior? This is the basic question behind our research plan, and our goal is to develop, dynamically, LOC (and other) models that better represent attribute behavior.

Observation and Comparison

To monitor the progress of a project, managers need

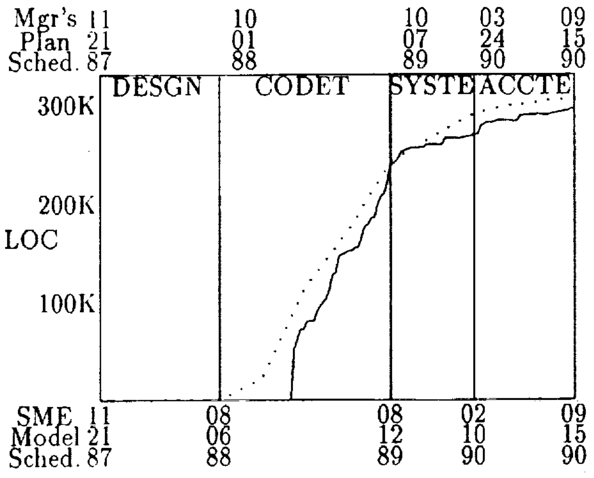

Figure 3: Growth in 'Lines of Code' for P_1

cumulative growth data for measures such as effort, size and errors. SME provides graphic display of the actual collected data like shown in Figure 3, in which the solid curve represents an overall view of project P_1's growth in size (lines of code) over a specified calendar time. The dotted curve in Figure 3 shows a LOC model of a similar project or the LOC measure model from the data base to permit the manager to compare project data to a model which indicates the "normal behavior" for such projects. Comparison can also be made between projects.

Prediction

SME can also predict a measure's completion value for an on-going project, by using the appropriate measure model scaled up to the actual time schedule of the new project. Using the initial data collected from a project, final values can be estimated giving the manager an indication of the measure's possible future behavior.

Analysis and Assessment

SME can help the manager identify the probable causes of any unexpected behavior for a measure, and assess the quality of a project based on all the measurement data. For each measure, a knowledge base of cause-effect relationships is maintained. So, for example, if a given project seems to have too many errors at a certain point in the coding phase compared to the error measure model, a rationale can be provided to the manager, such as:

```
TEAM IS REPORTING INCONSEQUENTAIL ERRORS
INEXPERIENCED DEVELOPMENT TEAM
POOR USE OF METHODOLOGY
COMPLEX PROBLEM
etc.
```

Similar idea can be found in [4]. What is desired is a mechanism whereas this knowledge base can be updated dynamically as projects evolve.

3 Cluster analysis

Cluster analysis is the technique for finding groups in data [7] that represent the same information model. Biologists and social scientists have long used it to analyze their data. Here, we use it to find similar measure patterns within the collected software development data.

Clustering was used previously in an early SEL study [3] in order to determine possible patterns in projects by clustering the modules that make up the project. The results were somewhat inconclusive due to large variances within small modules and the many different attributes that contributed to the single value that was clustered. In this current study, we try to separate out different attributes and study their effects over time. This gives greater precision to the data we are looking at and eliminates much of the variability found in the earlier study.

3.1 Clustering

As stated in section 2.2, a measure pattern can be represented by a vector. Clustering is a method to determine which vectors are similar and represent the same or similar physical objects. There are several clustering and modeling algorithms, including:

- *Euclidean distance.* Each vector represents a point in n-space. Points near one another are in the same cluster.
- *Cosine.* Each measure pattern represents a vector from the origin. The cosine of the angle between two vectors represents the similarly in their components and hence their closeness.
- *Optimal Set Reduction.* OSR generates, based on search algorithms and univariate statistics, logical expressions which represent strong patterns in a data set [2].

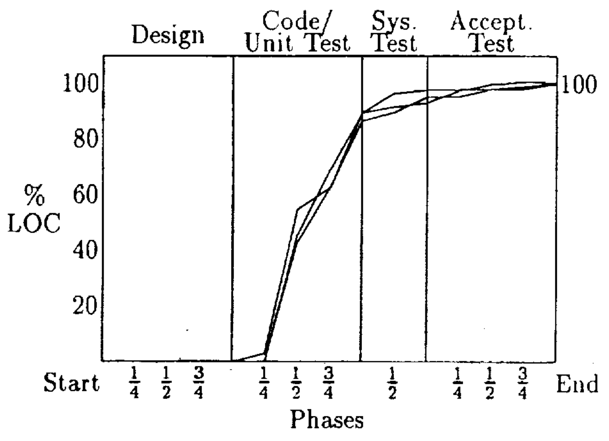

Figure 4: A cluster of LOC patterns

Several other algorithms also have been used.

For our initial investigation, we are using the Euclidean distance between two vectors as a degree of similarity between two measure patterns. For example, if $P = [p_0, p_1, p_2, \ldots, p_{13}, p_{14}]$ and $N = [n_0, n_1, n_2, \ldots, n_{13}, n_{14}]$ are two measure patterns, then their *Euclidean distance* is

$$ed(P, N) = \sqrt{(p_0 - n_0)^2 + \cdots + (p_{14} - n_{14})^2}$$

Two patterns are assumed similar and are in the same cluster if and only if $ed(P, N) < \varepsilon$.

Note that by varying ε we can adjust the size of the clusters by specifying how close two vectors must be in order to be in the same grouping. Since single vector clusters provide no information, we want to adjust ε so that we generally have clusters of at least 3 vectors without including vectors that represent fundamentally different curves. Figure 4 shows a cluster of three LOC patterns.

3.2 Cluster model

A *cluster model* is the average of all measure patterns in one cluster. It closely describes the measure behavior for all projects in the cluster because measure patterns in the same cluster are similar. Instead of choosing a predefined measure model for a project measure of interest using the project's characteristics (as is currently the case with SME), a cluster model can be dynamically selected for the project measure depending on which cluster its pattern best fits.

A further advantage from the current static approach of SME, is that alternative models can be developed for each measured attribute. Within SME, the same measure model is used for all measured attributes. For example, if the defining characteristic is Ada for the LOC measure, it will be the Ada measure model for each other measure (e.g., error, effort). With dynamic clustering, measure models can vary for each distinct measure.

For an ongoing project, a manager's estimate of schedule and measure completion values are used to derive its measure patterns. Estimates are replaced by real data once they become available. So a project measure's closest cluster model may change as the project develops. In Section 4.3 we discuss how to use this information to improve on the predictive capabilities of SME. On the other hand, since a project's development methodology or programming language usually do not change during a project's development life-cycle, the static measure model chosen by the current implementation of SME based on those characteristics does not change.

Similarly, SME does an assessment of a project's real data compared to the measure model's estimate by use of a predefined set of attributes. But by looking at the attributes that are common for all projects within a given cluster, we may be able to determine general characteristics for any new project that falls within that cluster. This list of attributes will dynamically evolve over time instead of being a static description of project behavior. For example, if all projects within a given cluster were previously late in delivery, it may be useful to report this information to the manager of a new project that falls within this cluster.

This allows the knowledge base to grow and change dynamically as projects develop. It does not require the predefinition of a few models - which may not even accurately represent the actual development model, only a manager's poor estimate of one.

4 Evaluation of Clustering

Before implementation of our clustering approach within SME, we evaluated the effectiveness of clustering with a subset of the SEL data base. Measurement data from twenty-four projects in the data base were clustered using eight different measures: computer hours (CPU), total staff hours (EFF), lines of code (LOC), modules changed (MCH), module

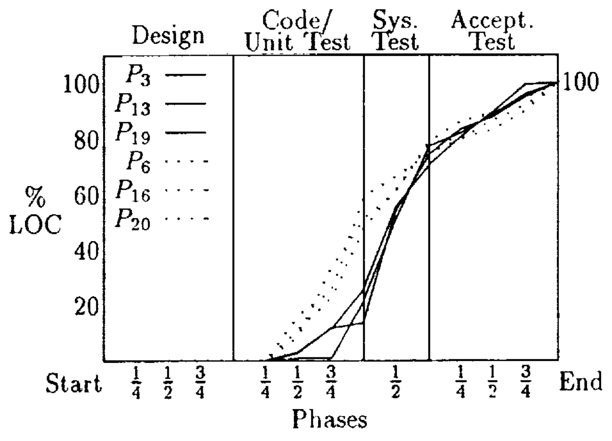

Figure 5: Two clusters of MCH patterns

Attributes	P_3	P_{13}	P_{19}
Computer	IBM	DEC	IBM
Language	FORT.	FORT.	FORT.
Application	AGSS	SIM.	ORBIT
Reuse (%)	10.1	30.7	38.1
Time (wks)	116	119	109
Size (SLOC)	178.6	36.6	15.5

Table 2: Project characteristics for cluster C_1

Attributes	P_6	P_{16}	P_{20}
Computer	IBM	IBM	IBM
Language	FORT.	FORT.	FORT.
Application	AGSS	AGSS	AGSS
Reuse (%)	19.5	1.9	10.0
Time (wks)	97	87	147
Size (SLOC)	167.8	233.8	295.4

Table 3: Project characteristics for cluster C_2

count (MOD), reported changes (RCH), reported errors (RER), and computer jobs (RUN). We then studied common objective and subjective attributes of projects in the same cluster.

For example, Figure 5 shows two clusters of MCH (module changes) patterns. Cluster C_1 consists of patterns from projects P_3, P_{13} and P_{19}, and cluster C_2 consists of patterns from projects P_6, P_{16}, P_{20}. We observe that more than half of the module changes were made during the code and unit test phase for projects in C_2 compared to about twenty precent for projects in C_1. Consequently, only twenty percent of the module changes were made during the system test phase for C_2 compared to about fifty percent for C_1.

4.1 Objective characteristics

Project characteristics of the two clusters are summaried in Table 2 and 3 respectively. We observe that if computer language is the basis for choosing a MCH measure model, as is the case with the current version of SME, all six projects will use the same MCH model since they all use FORTRAN. In this case, clustering discovers the two vastly different behaviors of MCH measures which are undetectable with the static approach.

In addition, some commonly used discriminators do not appear to be significant with these clusters. Size is often used to classify projects, yet cluster C_1 contains projects from 16K to 179K source lines. The projects represent two very different hardware and software environments (IBM mainframe and DEC VAX VMS) and each project in C_1 represents a different application area. (However projects in C_2 are more homogeneous; they all represent relatively large 168K to 296K attitude ground support systems built as mainframe IBM applications.)

4.2 Subjective characteristics

Subjective data for each project is stored in the data base as an integer between 1 (low or poor) and 5 (high). Each project manager fills in these values at the end of a project based upon experiences during the development. For each cluster we retrieved those subjective attributes that differed by at most 1 within the cluster, thus indicating a common feature for those clustered projects. This information can then be fed back to the manager of a new project that falls within that cluster to provide an indication of probable future behavior.

Projects in cluster C_2 have common ratings on the following subjective attributes:

```
Tightness of schedule constraints: 3
Access to development system: 3
Timely software delivery: 4
```

We notice that their rating for timeliness of software delivery is relatively high. This could be a direct result of the fact that most module changes were made during code and unit test phase.

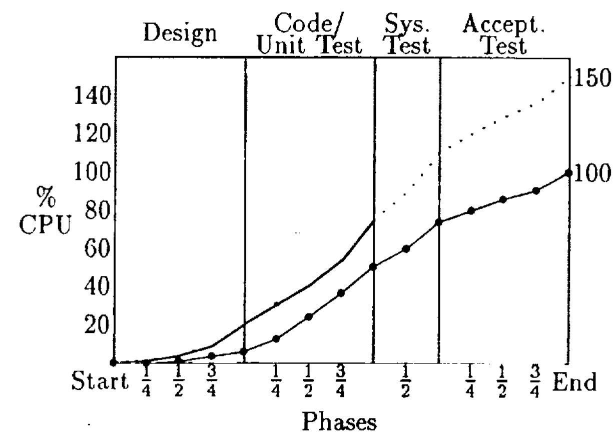

Figure 6: Prediction for CPU

4.3 Predictive models

The two clusters of Figure 5 are easiest to measure when all data points for each measure model are available. However, it is the very nature of predictive models that some of this data is incomplete. We are currently altering SME's predictive capabilities to take this into account.

If data is available for new project P up through point i (e.g., values for $p_0, p_1, \ldots, p_i$), then clustering for P against each existing cluster will be only with respect to these $i+1$ points. That is, for each cluster C, it will be assumed that p_i and c_i have the same value and P's other values will be scaled accordingly. Clustering will determine which cluster has the closest shape to P's shape.

Once a matching cluster is found, it will be assumed that project P has the same characteristics as this found cluster and the succeeding values for P will match the cluster's measure model for points $i+1$ through 14.

The effect will be to scale P's original estimate with respect to the cluster's estimate. For example, in Figure 6, if the cluster estimated a 50% completion by point 8 and the actual data showed a 75% "completion," then it can be assumed that the actual completion will be 150% since the relevant cluster is only half finished. In this case it can be assumed that the manager underestimated the resources needed for this project. We are currently modifying SME's graphical interface to show these predicted curves.

The predictive model for project P depends upon both estimating the total resources needed in order to compute the percentage for point P_i and estimating the schedule in order to determine how far one has progressed in the current development phase. Either one, however, may not be accurate. For example, current point P_6 represents 50% coding, yet that is only known when coding is complete. The current date may possibly range from perhaps the 25% level (and hence really represent point P_5) to the 75% level (and hence really represent point P_7) depending upon how accurately the initial schedule was set up. The true date will be known only after the coding phase is completed. However, in the above paragraphs we have described a mechanism to estimate resource needs when we assume that the schedule is correct.

On the other hand, if the latest available project point P_i is scaled to a cluster model horizontally along phases instead of vertically (i.e., by changing the estimated schedule), we can predict future changes in project schedule. However, since only discrete milestones of a schedule are used, they need to be quantified before numerical scaling can be applied. We are looking at extending the SME predictive model in order to estimate both the resource needs as well as potential bounds on the schedule based upon current data.

It should be realized that the model's predictive capabilities improve as a project develops. Very few points are available for prediction early in the development cycle leading to few differences among the various clusters. On the other hand, late in the development cycle where there is more variability among the clusters, it may be too late to change development models to account for any potential problems. How well the early predictions lead to significant differences in project development attributes is obviously an issue we need to investigate.

4.4 Model evaluation

Aside from its primary use as a tool to aid management in predicting future behavior on a current software development project, use of cluster analysis permits SME to be used as a tool to evaluate new models. If a model is proposed that describes some attribute of development that is collected by the SEL data base, then all projects within a cluster should exhibit that attribute to a great extent.

For example, the SEL is currently planning to enhance the SEL data base with additional predefined measure models in addition to the two models used at present. Often the following attributes (and their relevant values at NASA) are viewed as important attributes of a development methodology:

- *Computer use* – IBM or DEC environment
- *Reuse of existing source code* – Low, medium or high reuse of existing source code
- *Language* – FORTRAN or Ada as a source programming language
- *Methodology* – Cleanroom or standard NASA waterfall development method

By choosing one value from each category, the SEL can develop 24 possible models. A subset of these will be built into the SEL data base as predefined models for each project and each project will be assigned to one of these categories. However, while they are often viewed as crucial attributes, are these really discriminators useful to differentiate among projects?

If these are really discriminators of project development, then projects within a single cluster should all consist of the same predefined measure model (or at least predominately so). We can then use our clustering approach to determine the effectiveness of the new proposed models.

We can also use clustering to determine if there are any relationships among measures. If a cluster for Reported Change (RCH) consists of the same projects as a cluster for Reported Error (RER), this indicates that those two measures are closely related. If projects A and B are in the same cluster for CPU, LOC and RUN, then those projects are somewhat related.

This approach can be extended to any quantitative model. Projects in the data base can be grouped according to how well they meet the discriminators of any new proposed measure. The projects can be clustered, and if the models are appropriate, then clusters should be somewhat homogeneous.

For example, cleanroom is a technique that addresses early verification of a design that should result in fewer resulting errors (with less testing necessary) later in the development cycle. If so, then measuring reported errors (RER) per computer run (RUN) should cluster cleanroom projects together, and the plots should show high measure model values early in the development cycle. We can use SME to test such claims from this and other proposed models.

4.5 Evaluation of clustering

Clustering is effective in distinguishing measure behaviors. For most of the measures studied, we were able to yield clusters that differentiated behavior among the projects, whereas the current SME would consider them all similar and use the same measure model on that data.

A current weakness, however, is that the resulting clusters yield few common objective or subjective characteristics. We believe that this is due more to the nature of the current subjective files within the SEL data base than in the clustering methodology itself. The current data files are developed by the project managers and contain attributes about the project (e.g., external events such as schedule and requirements changes, team composition, environment composition). There is little about how management was performed (e.g., we didn't test enough, we started coding too soon). This is understandable given how the data was collected. We need to develop methods to collect this latter data in a non-threatening manner from each project manager so that it can be fed back to future project managers more effectively.

5 Conclusion

In this paper, clustering is presented as a mechanism for dynamically determining and altering the information model that describes certain attributes of the software development process. This permits the software manager to more accurately predict the future behavior of a given project based upon similar characteristics of existing projects in a data base. We believe the resulting cluster models are fairly accurate indicators of such behavior.

Clustering also permits rationale for deviations from normal behavior to be determined dynamically and are easier to generate than the existing expert system approach. Preliminary evaluation of clustering leads us to believe that the resulting models are fairly accurate indicators of such behavior.

In addition, it appears that some often used discriminators may not be totally effective in classifying projects. Size, programming environment and application domain may unnecessarily separate projects into categories that are ultimately the same (e.g., see

Tables 2 and 3). Obviously, this needs further study.

We are in the process of modifying NASA/GSFC's SME management tool for incorporation of these new models into the tool. We believe that this should greatly improve SME's predictive capabilities. Modification of the data in the SEL subjective data files should greatly aid in the analysis and assessment aspects of SME.

However, the process is far from over. We also intend to study alternative clustering and modeling techniques (e.g., Optimal Set Reduction, Cosine) in order to determine the best approach towards measuring these critical attributes. In addition, we need to observe how well early predictions of a project match with subsequent observations in order to be able to use SME as an effective management planning and tracking tool.

6 Acknowledgement

This research was supported in part by grant NSG-5123 from NASA/GSFC to the University of Maryland. We would like to acknowledge the contribution of Jon Valett of NASA/GSFC and Robert Hendrick of CSC as major developers of the original SME system and for their and Frank McGarry's (also of NASA) helpful advice on proposed changes we are making to SME.

References

[1] B. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.

[2] L. C. Briand, V. R. Basili, and C. J. Hetmanski. Providing an empirical basis for optimizing the verification and testing phases of software development. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering*, Research Triangle Park, NC, October 1992.

[3] E. Chen and M. V. Zelkowitz. Use of cluster analysis to evaluate software engineering methodologies. In *Proceedings of the Fifth International Conference on Software Engineering*, San Diego, CA, March 1981.

[4] R. Chillarege, I. S. Bhandari, and et al. Orthogonal defect classification – a concept for in-process measurements. *IEEE Transactions on Software Engineering*, 18(11), November 1992.

[5] W. Decker, R. Hendrick, and J. Valett. The software engineering laboratory (sel) relationships, models, and management rules. Technical Report SEL-91-001, The Software Engineering Laboratory, NASA Goddard Space Flight Center, Greenbelt, MD, February 1991.

[6] R. Hendrick, D. Kistler, and J. Valett. Software management environment (sme) concepts and architecture (revision 1). Technical Report SEL-89-103, The Software Engineering Laboratory, NASA Goddard Space Flight Center, Greenbelt, MD, September 1992.

[7] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, New York, NY, 1990.

[8] R. E. Park. Software size measurement: A framework for counting source statments. Technical Report 92-TR-20, Software Engineering Institute, Carnegie Mello University, Pittsburgh, PA, September 1992.

[9] A. A. Porter and R. Selby. Empirically guided software development using metric-based classification trees. *IEEE Software*, 7(2):46–54, 1990.

[10] R. Selby, A. Porter, D. Schmidt, and J. Berney. Metric-driven analysis and feedback systems for enabling empirically guided software development. In *Proc. 13th International Conference on Software Engineering*, pages 288–298, Austin, TX, May 1991.

[11] J. Tian, A. Porter, and M. V. Zelkowitz. An improved classification tree analysis of high cost modules based upon an axiomatic definition of complexity. In *Proc. 3rd International Symp. on Software Reliability Engineering*, Research Triangle Park, NC, October 1992.

[12] J. D. Valett. Automated support for experience-based software management. In *Proceedings of the Second Irvine Software Symposium (ISS '92)*, Irvine, CA, March 1992.

[13] A. von Mayrhauser. *Software Engineering: Methods and Management*. Academic Press, Inc., San Diego, CA, 1990.