A Comparison of Techniques for Extracting Information from Software Project Data

Roseanne Tesoriero
Department of Electrical Engineering & Computer Science
The Catholic University of America
Washington, DC 20064
202-319-5019
tesoriero@cua.edu

Marvin Zelkowitz
Department of Computer Science &
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742
and
Fraunhofer Center
for Experimental Software Engineering
College Park, MD 20742
301-405-2690
mvz@cs.umd.edu

Abstract

In order to compare a new software development to previous developments one usually has to characterize the existing environment and determine the characteristics governing that environment. Often a baseline model is built that is empirically determined by collecting data from a previous class of projects, and a composite model is built that represents the class of projects. However, if you can develop a theoretical model of how a baseline should behave and you collect baseline data from a set of projects, then you can determine whether the empirical baseline agrees with the theoretical model. In this paper we will use the WebME data visualization tool to investigate this question. We will look at various classes of data (e.g., effort data, error data) taken from the NASA Software Engineering Laboratory (SEL) and determine whether real project data agrees with the theoretical characterization of that data.

Keywords: Characterization, Measurement, Data collection, Visualization

1.0 Introduction

In order to compare a new software development to previous developments one usually has to characterize the existing environment and determine the characteristics governing that environment. How are resources spent over time? How are errors found and fixed? How are personnel used? These are all questions that must be answered so that the risk can be identified, i.e., deviations from that model can signify an event (both good and bad) that management must react to. Is this development on track? Must additional resources be added for testing? Must the schedule be lengthened due to slippage? Answering these questions gives management a better handle on how the new task compares to standard tasks from the past.

Often a baseline model is empirically determined by collecting data from a previous class of projects, and a composite model is built that represents that class of projects. This collective model is used as the basis for comparing all new developments. However, in order for this baseline model to be effective, it must represent the same set of assumptions and operating conditions as the new project being measured.

Collecting empirical data, however, has the danger that the collected projects do not truly reflect the same characteristics as a new project. However, if you can develop a quantitative theory of how a baseline should behave and you collect baseline data from a set of projects, then you can determine whether the empirical baseline agrees with this theoretical model. If this is so, then you have increased confidence that your model of the process agrees with reality, and you can more readily use this model in other situations.

Various methods exist for computing baseline data. A common approach is a regression model where an equation is developed that minimizes the error between a predicted value and an actual value. We can develop regression models for linear equations, quadratic equations, or any other predefined relationship that we wish. The danger here is that we do not really understand if the given data reflects that relationship, only that the set of collected data does fit the expected mathematical relationship.

A more sophisticated model would use a clustering relationship first to isolate subsets of data that have similar characteristics, and then use a regression approach to generate a best approximation to the underlying relationship. In this case, we have first identified separate categories of projects via the clustering algorithm. By now using a regression model, we have (it is assumed) chosen those projects that have similar development characteristics, so the resulting regression line more readily represents the underlying relationship.

As we discuss later, we have been working on an approach similar to this, but rather than using a common regression process, we previously developed a baseline characterization model called the *characteristic curve* that attempts to determine trend changes in the data. This signifies changes in the underlying development process, which provides a method to alter the underlying relationship during different phases of the development cycle. For example, measuring source code errors has a radically different growth characteristic during the design phase (with no source code to yield errors), during the coding phase (when source code errors occur often) to the testing phase (when new errors should be rare and existing errors start to disappear). This led us to the question of what are the underlying equations, if any, that govern such baseline characterizations?

In this paper, we will look at various assumptions that underlie baseline characterizations and we will try to correlate those assumptions with data collected from a variety of software development projects. Does this collective model meet any idealized norm? Do resource usage and reliability measurements agree with some underlying theoretical foundation? For example, in the late 1970s there was considerable interest in resource usage (e.g., effort data) and its relation to the Rayleigh curve ($y=2\ K\ a\ t\ \exp(-a\ t^2)$) as a formal effort model. In this paper we will investigate this previous work on formal development models for both resource usage and error data.

We will demonstrate this analysis using the capabilities of the tool WebME, explained in the next section, which can aid in this analysis. We show how a powerful plotting package can aid in analyzing baseline characteristics and presenting information back to the developer.

1.1 The Software Engineering Laboratory Environment

The data that we will present comes from the Software Engineering Laboratory (SEL) at NASA Goddard Space Flight Center. The SEL was organized in 1976 to study flight dynamics software, and since that time it has had a significant impact on software development activities within the Flight Dynamics Branch (since January 1998, the Information Systems Center). Many technologies have been studied (e.g., defect analysis, resource usage, Ada, cleanroom, IV&V, COTS, OO design) and have been reported elsewhere [2].

As a brief overview of SEL operations, the SEL has collected and archived data on over 150 software developments. The data are also used to build typical project baselines against which ongoing projects can be compared and evaluated. Projects range in size from approximately 10K lines of source code to 300K to 500K at the high end. Projects involve from 6 to 15 programmers and typically take from 12 to 24 months to complete. FORTRAN was the original development language, with a movement to Ada in the late 1980s and then to C and C++ in the 1990s.

1.2 Data visualization

In order to aid in understanding the collected data, the SEL has developed two data visualization tools. The Software Management Environment (SME) [4] provided quasi-real-time feedback on project data. Data would be entered in a database within two to three weeks of it being collected, and then a program was run to extract that data for entry into the SME database. Management could then use SME to display growth rates of certain project attributes (e.g., lines of code, staff hours, errors found) and compare them to previous projects with similar characteristics.

The weakness of SME, however, was that knowledge of software development is built statically into the tool. That is, certain characteristics of the baseline model were inherent in the design of SME. For example, only two development models were included -- the large attitude systems written in Fortran and the smaller onboard simulators written in Ada. All projects were classified according to either of these models, even it if didn't fit those parameters. While appropriate for the SEL environment of the early 1980s, it didn't fit the changing role of flight dynamics software in the late 1980s and 1990s.

To alleviate this problem, a second tool, WebME (Web-based Measurement Environment) was built to dynamically baseline a set of previous projects that "look like" a given development in order to characterize a new project [6][1]. This collective model is used as the baseline for comparing all new developments. In the analysis that follows, we use WebME as our tool for processing the models we wish to investigate.

2.0 Calibration using effort data

As a first step, we first calibrated our process using WebME on effort data. Tracking resource expenditures is a relatively standard process by many organizations and a fairly extensive literature exists to compare our results against.

The SEL collected effort reports as hours of activity per week on each project. An important concern is to

[1] WebME has other characteristics, principally the ability to collect data across a distributed development environment using the world wide web. However, the dynamic modeling of baseline data is of primary concern in this paper.

understand if the effort on a project is tracking according to preconceived plans. However, real-world data is generally characterized as noisy. For example, Figure 1 represents the staff hours by week for one particular SEL project. As shown, there seems to be little structure in the data.

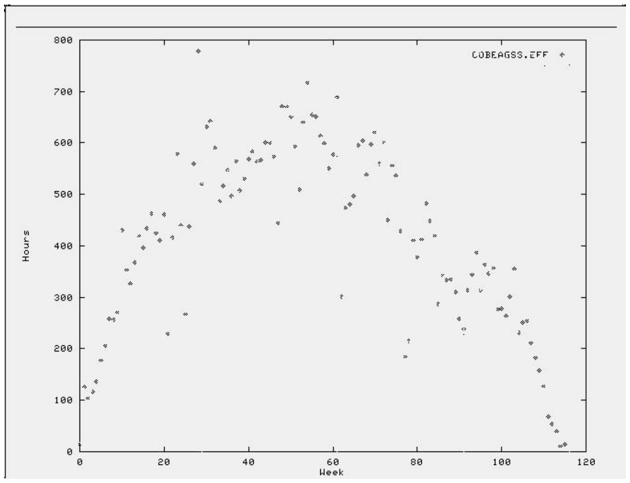


Figure 1. Resource data as hours per week

An important goal for baseline characterization is to determine whether there is an underlying model that structures this data. We can apply various smoothing techniques to achieve some structure. For example, Figure 2 represents 3 methods for determining the structure for the data of Figure 1:

We can compute a moving average (e.g., average of successive data points). This causes local perturbations to disappear. The jagged line of Figure 2 represents this approach.

We can compute an approximation of the data. Earlier we developed a model called the characteristic curve [5]. The 9-section segmented line of Figure 2 represents the results of applying this approach. In this case we try to determine trend changes in the data, as signified by changes in the line segments. We call these points *pivot points*, and claim that they represent changes in the underlying model of the data. In this case we are building a formal model empirically by looking at the data itself.

In both of the above 2 cases, however, we are creating a baseline object that is based upon the empirical information that is present. Work in the 1970s identified the Rayleigh curve

$$y=2\ K\ a\ t\ \exp(-a\ t^2)$$

as an approximation of resource usage on software development projects [1]. The theory of the Rayleigh curve grew out of hardware reliability theory. The effort needed to work on a software development is proportional to the number of modules that still need to be written. But this is related to the number of modules that already exist. The solution to this is a second order differential equation with the Rayleigh curve as its solution. It is often given in its integral form as:

$$\text{Cost} = K\ \exp(-a\ t^2),$$

with periodic resource expenditures as its derivative.

This curve has 2 parameters: *K*, which is the area under the curve (i.e., the entire cost of the project in hours) and *a*, which represents the *skewness* of the curve. If T_d represents the time at which the curve is maximal, then $a=1/\ T_d^2$. If we normalize the area under the curve to be 1 (i.e., the curve represents a probability distribution), then each point represents the percent of effort that occurs during a given time period. We have then reduced resource use to the single parameter, *a*, which is only a function of the time when expenditures reach a maximum[2].

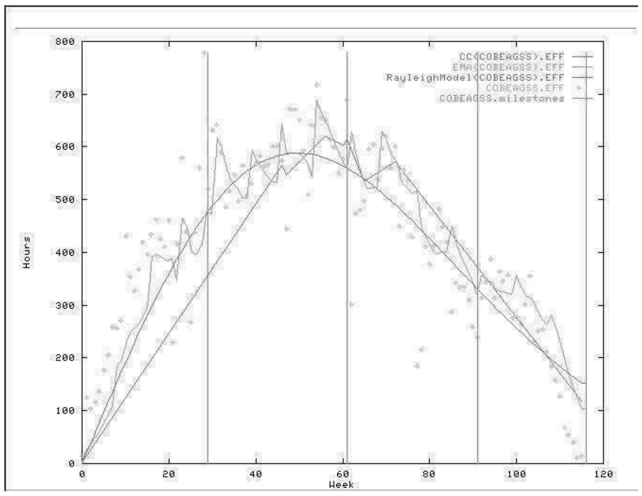


Figure 2. Modeled resource data

The question that needs to be asked is which of these models (or any other you care to develop from the original data) represents the "true" meaning of the scatterplot of Figure 1? As Figure 2 shows, all 3 are reasonable approximations of the data. However, each has a different interpretation.

[2] An interesting sidelight to this study is that in our original 1978 Rayleigh curve analysis [1], it took approximately 12 hours to analyze and plot the Rayleigh estimators. In this current study it took about 1 hour to analyze and prototype the Rayleigh model for WebME, about 30 minutes to write the WebME instrument (executable program) to plot the points and 15 minutes to write the script to add the model to WebME. Each graph now takes a single menu click and about 10 seconds to plot.

The **average curve** represents reality for this particular project. It simply records what has happened. There is no way to determine whether this can be considered a normal or abnormal plot. We can collect several projects and compute an average of those as a baseline. We can compare the original project average and this baseline only if we can be sure they represent the same project characteristics.

The **characteristic curve** represents certain decision points in the data. It is a best-fit set of line segments whose end points (i.e., pivot points) represent local minima and maxima (i.e., zero first derivatives in the original data). These indicate trend changes. Unlike the above averages baseline, the use of derivative trend changes imposes a structure (albeit an assumption that we are making in how the model behaves) on the data that is missing from the previous averages model. Whereas the averages baseline is purely driven by the data that is present, the use of the characteristic curve imposes a structure on the meaning of the data.

Therefore, this class of model is more of a *descriptive or evaluation model*. Collecting a baseline of similar projects has more meaning than before. If we again collect several projects and compute the characteristic curve of the average among this collection, differences between the two indicate different trends and indicate different operating characteristics between the set of baseline projects and the new project.

The **Rayleigh curve** is the only estimator of the three that is firmly based upon a theoretical model independent of the actual data. The computation of T_d is the only computation needed to fit the curve to the data. This is more of a *prediction model* without the need to first compute a baseline. We can use the Rayleigh curve to predict resource use into the future. If we build a baseline from a set of projects and it agrees with the Rayleigh estimator, then we have an empirical validation that the Rayleigh curve can be an estimator of resource expenditures. In addition, the use of a theoretical model allows us to build a “theoretical baseline” before we have a collection of projects to study. That is, we can compare a new development to this theoretical model before we have several projects to compare it to. This would be useful in the current SEL environment as the old Flight Dynamics Division has given way to the newer Information Systems Center.

Using WebME we evaluated the effectiveness of the Rayleigh curve as a predictor of resource usage. Figure 3 represents this data. It plots the Rayleigh curve estimator of Figure 1 with the average of the 8 projects[3] that WebME’s cluster analysis determined to be in the same baseline class. As shown, both curves have similar profiles, which gives credence to using a Rayleigh curve estimator.

[3] For this analysis, we are using WebME with a subset of the SEL database which includes 18 projects.

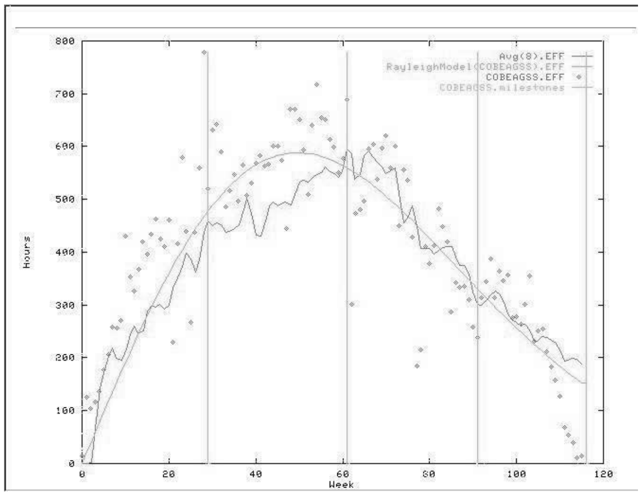


Figure 3. Baseline data

The original SME implementation used a static definition for baseline projects, which reduced to simply the implementation language as the discriminator. Large ground support systems were all written in Fortran and smaller on-board simulators were all written in Ada. Therefore only two baselines were built. For WebME we dynamically created baselines by looking at similar projects, based upon the data. A clustering algorithm was developed [3] and all projects with Euclidean distances closest to the given project were part of the baseline.

It should be mentioned that the algorithm we used ensures that all baseline projects look somewhat like the original project, so it is not too surprising that the average for the baseline is somewhat like the average for a single project, and hence similar to the Rayleigh estimator. However, the degree of similarity to the Rayleigh curve still makes the equation a good estimator. In fact, we obtained the following "distances" from our clustering algorithm for the following estimators for the project in Figure 3 and for 3 others in our database. (Numbers represent average separation between the estimating baseline curve and the original data, in hours per week for the effort data of the following table):

Estimator	Fig. 3	Proj A	Proj B	Proj C[4]
Moving average of all raw data points	93.9*	117.9+	64.8*	36.4
Characteristic curve	98.5+	107.9*	71.3+	30.0*
Rayleigh model	94.3	117.2	66.4	55.3+

* Best fit
+ Worst fit

All three models give approximately the same precision, with the Rayleigh estimator generally falling between the other two.

In Figure 4 (a-c) we apply the same Rayleigh estimator and clustering baseline on 3 different projects. We get reasonable results, except for projects where effort peaks later in the development cycle (Figure 4(c)). In this case, the Rayleigh curve underestimates resource use. Apparently a different process is happening on these projects, which requires further study.

In each case the baseline curves are dynamically developed by clustering those projects that have a similar development cycle. Figure 4(a) baseline includes 9 projects, as does Figure 4(b). The Figure 4(c) baseline only includes 7 similar projects.

3. Size characteristics

3.1 Estimating size

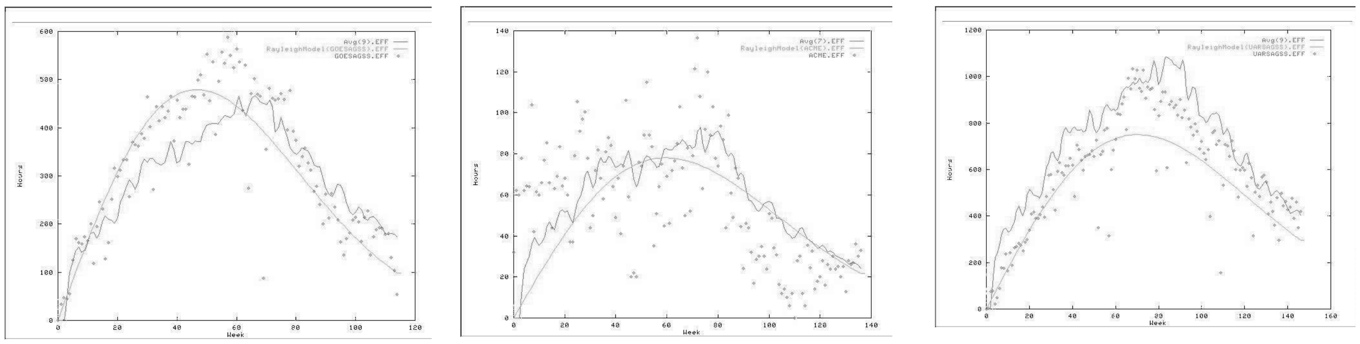
After studying effort data, we moved our attention to size data, principally lines of code. In looking at the assumptions behind the Rayleigh curve as an effort estimator, we believe that the same set of assumptions govern program size. That is the number of lines of code (e.g.., modules) that can be written at a given point in time depends upon how many other lines of code already exist. This leads to the same solution as in the previous effort estimators.

In order to test this hypothesis with WebME, we had to make a simple change, however, to our WebME Rayleigh curve model. In most developments, source code production does not begin until several time units (e.g., months) have passed. We modified the model to ignore those initial zero points and to estimate T_d and a once source code production started.

Using this modified model, we plotted several projects, giving both the Rayleigh estimator and the moving average (Figures 5(a) and 5(b)). As shown, the Raleigh curve remains a reliable estimator of code production on a project, and could provide a reasonable estimate on overall project size that can be compared to other estimators that a project manager may use.

3.2 Productivity

Program size is not the only size estimator of use to project management. Measuring productivity provides another measure of overall performance. However, most modeling of productivity hides important characteristics hidden in the data. For example, Figure 5 shows lines of code production by week. We can, in Figure 6(a) exhibit productivity by week by dividing by the effort for that week. However, each weekly total by itself does not shed much light on the process. In Figure 6(b) we plot the growth of a project each week. In this case we can compute productivity plotting the total size of the project by week and then by dividing the size of the project by the total effort to date. However, this too hides details of the



(a) (b) (c)

Figure 4. Rayleigh curve vs. Baselines on 3 projects

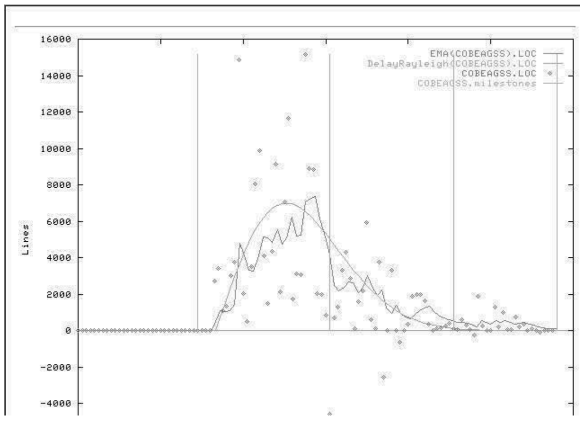
[4] As we explain for Figure 8 later, the poor fit for the Rayleigh curve here can be explained by the problems that this project had in finding and fixing defects.

process since it appears to be mostly a monotonically growing curve.

We solved this in WebME by plotting Figure 6(b) normalizing effort by week. That is, we plotted total system size in lines of code divided by total effort on a weekly basis. Figure 7 shows the results of this where we can see overall productivity on a week by week basis.

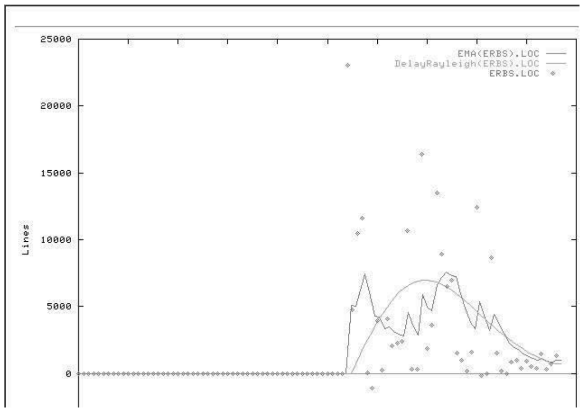
An entirely new attribute now shows up in Figure 7. We see that productivity remains 0 until week 33, when it rises rapidly. This corresponds to the start of the coding phase. Maximum productivity of 5.04 lines per hour is reached in week 57, where it peaks and then starts to drop. It drops to a point of 3.075 lines per hour in week 115, which represents the end of the project.

This clearly shows that week 57 is a transition point in the project. Maximum source code production is reached in week 57 and the project enters the testing phase where generation of new lines of code is secondary to removing defects. Corroborating this are the facts that week 56 is a pivot point for effort (Figure 2) with a maximal value of 621 hours of effort and week 57 is a maximal pivot point for source code production (not shown) of 7021 lines of code produced. By developing this weekly productivity estimator with WebME from the data for this project, we were easily able to show changes in the development behavior in this project.



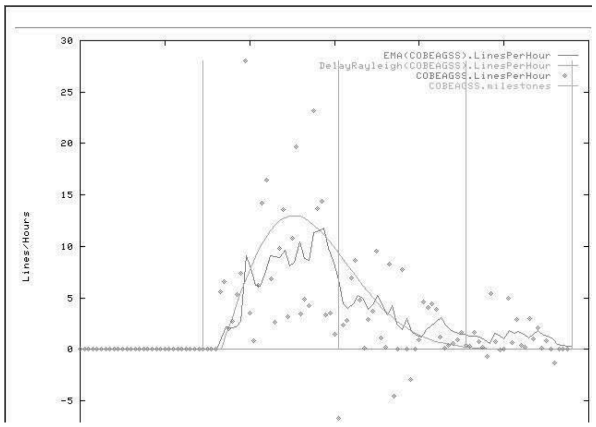
(a) Project A

Figure 5. Lines of Code estimators

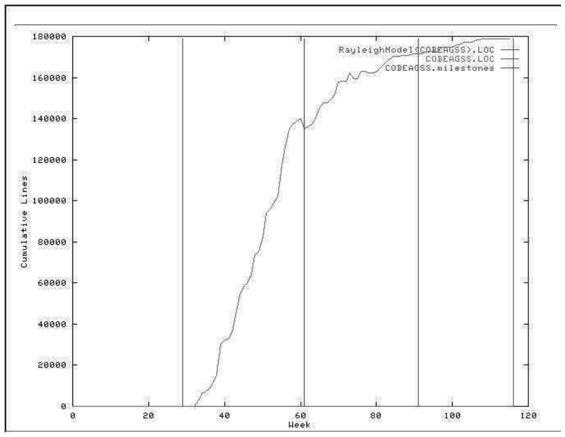


(b) Project B

Figure 5 (cont). Lines of Code estimators



(a) Lines/hour by week



(b) Total lines by week

Figure 6. Productivity

The slope of the line in Figure 7 from weeks 57 through 115 is negative productivity and shows the effort expended in testing and fixing defects. Any testing phase will by necessity incur some loss in productivity, which management should monitor. The slope of this line is a predictor of poor performance.

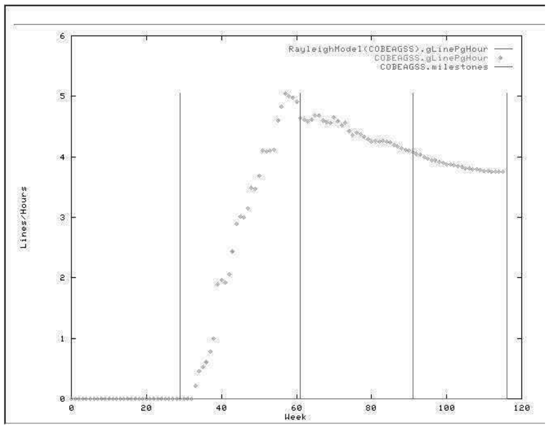


Figure 7. Productivity by week

Not every project, however, has the "cliff” shape of Figure 7. Figure 8 presents the productivity from another project. Note that there are productivity "dips" throughout the second half of the project. This means that there were significant time periods throughout the process where new code production stopped and defect repair took a major effort. Rather than coding throughout the coding phase, significant time was spent in repair and redesign, probably due to poor initial design. Looking at defect rates for this project shows that some of this assumption is true.

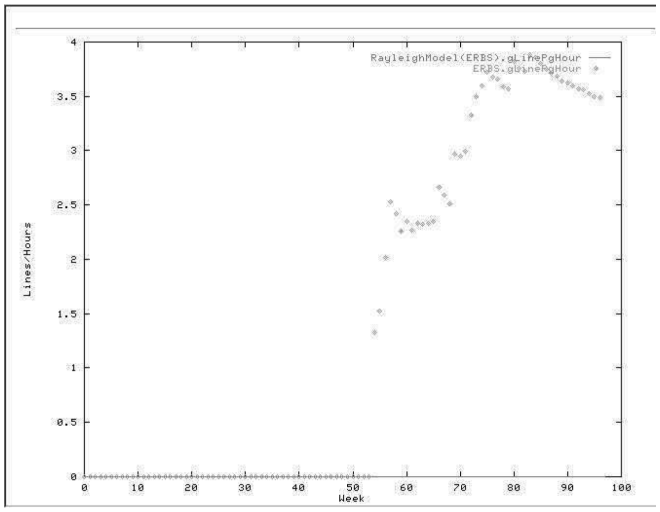
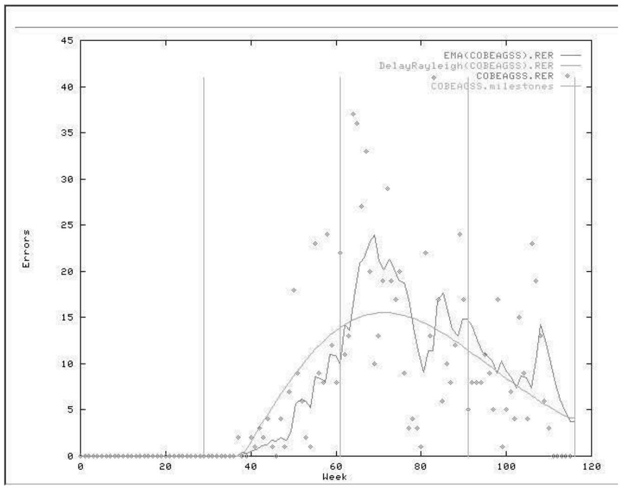
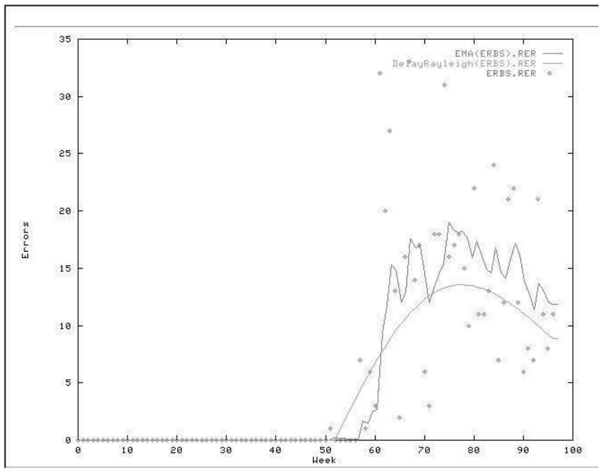


Figure 8. Less structured project

Figure 9(a) represents a typical SEL project. Here we plot both the moving average and Rayleigh estimator of defects. In this case the Rayleigh estimator is approximately the same as the moving average. Thus the real data (e.g., defect rates) agrees fairly closely with the theory. Defects are found fairly early in the testing process and reach a peak in week 69; about 3 months after the end of the coding phase in week 57. As with effort and size data, the Rayleigh curve is again a reasonable estimator of defect rates.



(a) **Defect rates from Figure 7**



(b) Defect rates from Figure 8

Figure 9. Defect Rates

However, the defect rates of Figure 9(b) show the same chaotic behavior of Figure 8. The moving average has a greater value than the Rayleigh estimator later in the development cycle. In this case, weekly defect rates reach about 15 a week by week 63 and remain at that level until week 89, a span of 6 months. It seems like the problem would have been more apparent earlier in the development cycle by reporting data as in productivity measures as in Figure 8 rather than as simply error counts as in Figure 9.

4. Conclusions

In this paper we have looked at the process of building models of development data according to three criteria:

1. **Basing models upon the data itself**. The moving average is a common approach that is used. This model provides no predictive capabilities; it is simply an accounting of what happened. Predicting using this model depends upon choosing projects with the same characteristics; something we only know how to do poorly.

2. **Basing models on empirical characteristics of the data**. We previously developed the characteristic curve as a means of determining trends in the data. This imposes some structure on the data that is absent from the purely empirical approach. The characteristic curve, for example, is less precise than the moving average, but its implied assumptions allow for interpreting the data from other projects more effectively than with the moving average model.

3. **Basing models on a theoretical foundation**. We have been using the Rayleigh curve from hardware reliability theory as an underlying model for software growth. This model provides less information about the specific project, but allows the model to be used as a predictive model of future behavior.

Basing models purely on the data lacks a predictive nature to the characterizing baselines. However, the latter two techniques allow us to evaluate and predict future behavior with some degree of precision. We have found that the Rayleigh curve seems appropriate in our environment.

Displaying productivity data, as given in Figures 7 and 8 as project productivity by week, allows us to view characteristics of the project that are not readily apparent in other ways. In particular, the shape (or lack thereof) of the productivity curve in Figures 7 and 8 indicate potential problems before other data indicates the same problem. The slope of this curve (negative productivity) indicates the amount of testing in a project, and further study to indicate the relationship of this slope to overall testing costs and length seems warranted.

Finally, this analysis could not have been accomplished without a tool such as WebME. The ability to easily develop models of a process and then have those models applied to any given project within a matter of seconds allows project management to easily compare a given project to other projects in a system database as a real-time project management tool. It is this instant feedback that is often lacking with other metrics and data collection activities. Tools like WebME could provide the inspiration to get management "on board" and agree to collect relevant data. This is often the hardest part of metrics analysis -- getting management "buy in" to the process. The easy interface to the database and near real-time analysis may be all that is necessary.

Acknowledgment

This research was supported in part by NASA grant NCC 5-170 from NASA Goddard Space Flight Center to the University of Maryland.

Bibliography

[1] Basili V. R. and M. V. Zelkowitz, Analyzing medium scale software development, Third International Conf. on Software Engineering, Atlanta, Ga. (May 1978) 116-123.

[2] Basili V., M. Zelkowitz, F. McGarry, J. Page, S. Waligora, and R. Pajerski, SEL's software process-improvement program, *IEEE Software* 12, 6 (1995) 83-87.

[3] Li N. R. and M. V. Zelkowitz, An Information Model for Use in Software Management Estimation and Prediction, Second International Conf. on Information and Knowledge Management, Washington, DC, (November 1993) 481-489.

[4] Hendrick R., D. Kistler, and J. Valett, Software management environment (SME) components and algorithms, NASA/GSFC Technical Report SEL-94-001, (February 1994).

[5] Tesoriero R. and M. V. Zelkowitz, A Model of Noisy Software Engineering Data (Status Report), International Conf. on Soft. Eng., Kyoto Japan, (April 1998) 461-464.

[6] Tesoriero R. and M. Zelkowitz, WebME: A web-based tool for data analysis and presentation *IEEE Internet Computing* 2, 5, (September 1998) 63--69.