Computers & Structures Vol. 10, pp. 39–43
© Pergamon Press Ltd., 1979. Printed in Great Britain

0045-7949/79/0401-0039/$02.00/0

MEASURING SOFTWARE DEVELOPMENT CHARACTERISTICS IN THE LOCAL ENVIRONMENT†

VICTOR R. BASILI and MARVIN V. ZELKOWITZ

Department of Computer Science, University of Maryland, College Park, MD 20742, U.S.A.

(*Received* 12 *May* 1978)

Abstract—This paper discusses the characterization and analysis facilities being performed by the Software Engineering Laboratory which can be done with minimal effort on many projects. Some examples are given of the kinds of analyses that can be done to aid in managing, understanding and characterizing the development of software in a production environment.

INTRODUCTION

Software development is big business. Estimates on the actual expenditures for software development and maintenance were ten billion dollars in 1973[1] and most likely 15–25 billion dollars today. These are only estimates because little data is gathered by the software industry in monitoring itself, analyzing its environment and defining its terms.

The software product and its development/maintenance environments cover a wide range. The product varies from first time, one of a kind systems, to standard multi-level run of the mill systems; from large scale hundreds of man–year developments to small scale one to two man–year developments. The environment varies from shops dedicated to the development of software to organizations which simply maintain their existing software system. A large number of methodologies, tools and techniques are available to help in the cost effective production of maintainable software. However, most of these techniques involve tradeoffs when applied in actual practice; some tools are impractical in certain environments and some techniques may not be applicable in other environments.

For example, for a new one-of-a-kind project where some specifications are still unknown or subject to change (not a recommended procedure), incremental development techniques, such as iterative enhancement[2] may be more cost effective than the more standard top down approach. Some tools, such as requirements analyzers[3] which are highly effective in the development of large scale systems, are not effective when the project is relatively small due to the substantial overhead in using the tool. Peer code reading is impossible in an environment of only one programmer.

Understanding the characteristics of a particular software environment leads to more cost effective maintainable software. This requires knowledge of the various parameters that affect the development of software and its maintenance. Unfortunately there is little effort expended in analyzing this process in local environments. Most of the data has come from the very large scale developments, projects like OS360, Sage, Gemini and Saturn[4].

Although these projects are major contributors to the software development budgets, they are not necessarily typical of software development across the industry. However, they are easiest to secure funding for collecting data and analyzing it. For example, if the budget for a project is twenty million dollars, then it is easy to add two hundred thousand for data collection and analysis, a mere 1% overhead. However, if the project has a budget of two hundred thousand dollars, then adding fifty thousand for data collection imposes a prohibitive 25% overhead.

What characterizes these large scale software development projects? The development activities usually involve about 30% analysis and design, 20% coding and 50% testing. However, development costs account for only 20% of total system costs on some projects if maintenance and modification activities are included[1].

These cost characteristics however are different for different software environments. What characterizes the projects studied above is that they are large one time only systems. Testing is very expensive because it is difficult to integrate the various pieces of the system into a working unit. Clearly smaller better understood systems would require a smaller proportion of the testing time and possibly less design and analysis time.

The authors have been analyzing development in an environment in which the software is of the six–ten man–year variety involving the development of ground support software for spacecraft control; a set of problems whose basic solutions and designs are fairly well understood. Thus the tailoring of methodoligies and tools for this environment would surely be different than in other environments.

THE SOFTWARE ENGINEERING LABORATORY

The Software Engineering Laboratory began in August, 1976 to evaluate various techniques and methodoligies to recommend better ways to develop software within the local NASA environment. Three groups participate in the Laboratory—the University of Maryland, whose role is to develop an operational measurement environment and analyze the development process; NASA Goddard Space Flight Center, whose role is to implement the operational measurement environment and whose goal is to discover ways to develop more product for the money spent; and the contractor, Computer Sciences Corporation, whose role is to supply data as they develop software and whose goal is to gain feedback on project development both for understanding

†Research supported in part by grant NSG-5123 from NASA Goddard Space Flight Center to the University of Maryland.

the characteristics of past development and to monitor software development in real time.

More specifically, the goals of the Laboratory are:

1. Organize a data bank of information to classify projects and the environment in which they were developed.
2. Compare what is happening with what is supposed to be happening (e.g. are the proposed methodologies being employed as they are supposed to be implemented?).
3. Isolate the significant parameters that characterize the product and the environment.
4. Test out existing measures and models as they appear in the literature (usually for large scale software developments) and develop measures for the local environment.
5. Analyze methodologies and their instrumentation in the local environment.
6. Discover and recommend appropriate milestones, methodologies, tools and techniques for use under given conditions in order to develop more manageable, maintainable, reliable, and less expensive software products.

The research objectives of the Laboratory can be divided into three basic areas: management, reliability and complexity. The *management* study is to analyze and classify projects based on management parameters, and investigate management measures and forecasting models. The *reliability* study is to examine the nature and causes of errors in the environment, find classification schemes for errors and expose techniques that reduce the errors that occur in the local environment. The purpose of the *complexity* study is to gain insight into the nature of complexity and develop models that correlate well with those insights and discover whether various techniques create more systematic and thus easier to maintain program structures.

The primary data gathering technique for the Laboratory is a set of seven reporting forms:

General Project Summary

This form is used to classify the project and is used in conjunction with the other reporting forms to measure estimated vs actual project progress. It is filled out by the project manager at the start of the project, at each major milestone, and at completion. The final report should accurately describe the system development life cycle.

Programmer/Analyst Survey

This form is filled out by each programmer at the start of the project, and is used to classify the background of all project personnel.

Component Summary

This form is used to keep track of the components of a system. A component is a piece of the system identified by name or common function (e.g. entry in a tree chart, COMMON block, subroutine). With the information on this form combined with the information on the component status report, the structure and status of the system and its development can be monitored. This form is filled out for each component at the time that the component is identified (usually during the design stage), and at the time it is completed (usually during testing). It is filled out by the person responsible for the component.

Component Status Report

This form is used to keep track of the development of each component of the system. The form is turned in weekly by each person on the project, and it identifies the components worked on, hours devoted to each component, and tasks performed (e.g. design, code, review).

Resource Summary

This form keeps track of project costs on a weekly summary basis. It is filled out by the project manager and lists for all personnel the total number of hours spent on the project.

Change Report

The change report form is filled out every time the system changes because of change or error in design, code, specifications or requirements. The form identifies the error, its cause and other facets of the project that are affected.

Computer Program Run Analysis

This form is used to monitor computer activities used in the project. Entries are made every time a run is submitted for processing. The form briefly describes the purpose of the run (e.g. compile, test, file utility), and the results (e.g. successful, error termination with message).

DATA COLLECTION ON A SMALLER SCALE

The research goals of the Software Engineering Laboratory require the collection of large amounts of data to make full investigations into the nature of the software development process. The information being collected by the Laboratory, due to its research nature, is ambitious and not cost effective for simple management control; it requires a major expenditure just for processing and validating data for inclusion into the data base.

However, it is possible to gather less data to get effective results in analyzing the characteristics of the local software environment. For example, a subset of the information contained essentially on only three basic forms is used for the analysis in the next section. The three forms are the General Project Summary, the Resource Summary and the Change Report forms.

From the General Project Summary the following information is used:

1. Project description including the form of input (specifications), products developed and products delivered.
2. Resources of computer time and personnel, including constraints and usable items from similar projects.
3. Time including start and end dates and estimated system lifetimes.
4. Size of project including various measures such as lines of code, source lines and number of modules.
5. Cost estimates, man–month estimates and schedules.
6. Organization factors, personnel and the kinds of people used (e.g. managers, librarians, programmers).
7. Methodologies, tools and techniques used.

Data from the Resource summary includes weekly charges for manpower and computer time, and other costs for all categories of personnel. The change report form supplies data on changes made to the system, when they were made, what modules were affected by the change, and why the change was made.

PROGRESS FORECASTING

One important aspect of project control is the validation of projected costs and schedules. A model of estimating project progress has been developed and with it estimates on project costs can be predicted.

The Rayleigh curve has been found to closely resemble the life cycle costs on large scale software projects [5, 6]. The curve yielding current resource expenditures (y) at time (t) is given by the equation:

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

where the constant K is the total project cost, and the constant a is equal to 1/(Td**2) where Td is the time when development expenditures reach a maximum. The following analysis demonstrates how this data can be used for management control of a project. The data was obtained on projects built for NASA and monitored by the Software Engineering Laboratory.

For each project in the NASA environment, requirements analysis yields estimates of the total resources and development time needed for completion, which is recorded on the General Project Summary form. The following three parameters are relevant to this analysis:

1. Ka, total estimated resources estimated to be needed to complete the project through acceptance testing (in hours).
2. Yd, the maximum resources estimated to be needed per week to complete the project (in hours).
3. Ta, the number of weeks estimated until acceptance testing.

Since the Rayleigh curve has only two parameters (K and a), the above system is over specified and one of the above values from the General Project Summary can be determined from the other two. Thus the consistency of those estimates can be validated. Alternatively, by estimating two of these parameters (e.g. total cost and maximum weekly expenditures), then the third value (e.g. completion date) can be calculated.

For example, since budgets are generally fixed in advance, there is usually little freedom with total resources available (K). Also, since a fixed number of individuals are usually assigned to work on the project, the maximum resources Yd (at least for several months) is also relatively fixed. Therefore, the completion date (Ta) will vary depending upon K and Yd.

As stated above, Ka is the total estimated resources needed to develop and test the system through the acceptance testing stage. For each environment, the actual resources K must be obtained from this figure. There are several methods for estimating K. One approach is by the empirical data available on past projects. By studying past projects as NASA, this figure is 12% greater than estimated expenditures (hence K = Ka/.88). The remaining 12% is for last minute changes after acceptance testing. Since maintenance costs are not covered, this figure seems quite low when compared to other programming environments—the corresponding figure in other organizations that do include maintenance costs will probably be correspondingly higher.

Give K, a was computed by assuming different values of Td to yield the given value of Yd on the General Project Summary. Then given constant a, the estimated date of acceptance testing Ta can be comput as follows:

The integral form of the Rayleigh curve is given by:

$$E = K(1 - \exp(-at^2))$$

where E is the total expenditures up to time t. From the previous disucssion, we know that at acceptance testing, E is .88 K (for NASA). Therefore,

$$.88\,K = K(1 - \exp(-at^2)).$$

Solving for t yields:

$$t = \mathrm{sqrt}(-\ln(.12)/a).$$

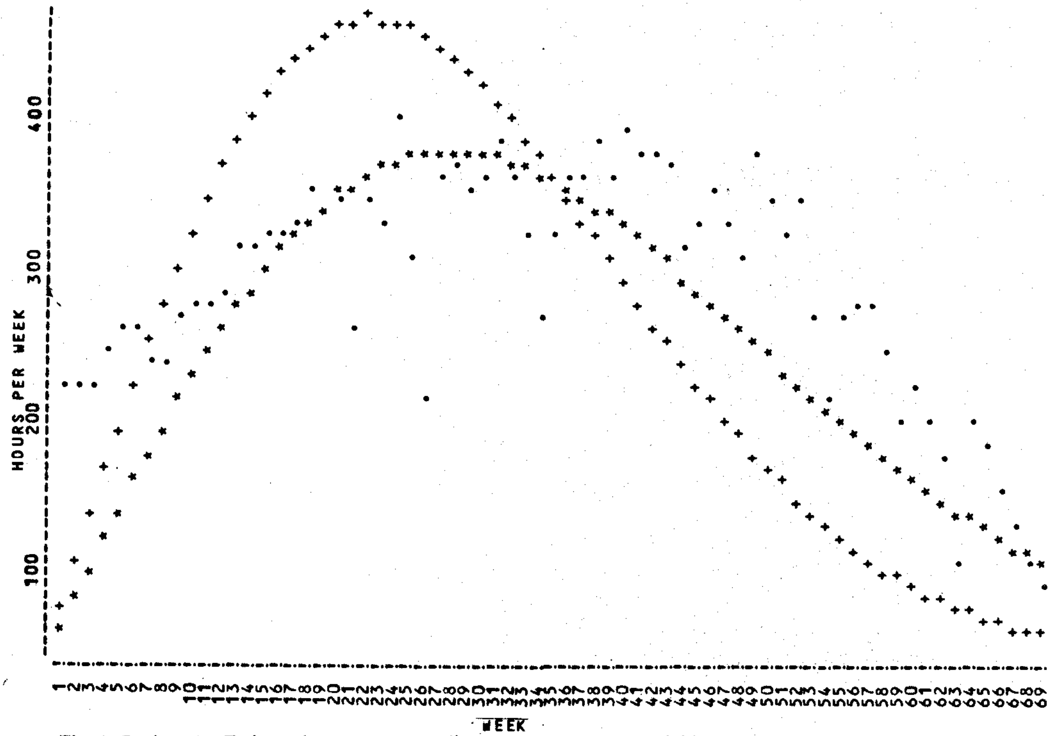


Fig. 1. Project A—Estimated resource expenditures curve based upon initial estimates. *, Estimating curve with Yd (maximum resources) fixed. +, Estimating curve with Ta (Completion date) fixed. ·, Actual data.

Also, in a second analysis, the estimated acceptance time Ta was fixed in order to yield a value of a (and hence Yd) that represents the manpower needed to finish on schedule.

If the original estimates from the General Project Summary are accurate, then the estimated and calculated values should be comparable. If the maximum manpower estimate was reasonable, then the predicted date for acceptance testing should be similar to the estimated date on the General Project Summary. If this acceptance date is reasonable, then maximum manpower estimates should be similar to the calculated values.

Figure 1 represents data from one actual project. According to the above analysis two different Rayleigh curve estimates were plotted. The curve limiting maximum weekly expendiutres (Yd) might be considered the more valuable of the two since it more closely approximates project development during the early stages of the project. In this case, the weekly expenditures from the General Project Summary were insufficient for completing acceptance testing by the initially estimated completion date Ta. The model predicted acceptance testing in 58 weeks instead of the proposed 46 weeks. The actual date was 62 weeks—yielding only a 7% error (Fig. 2).

In order to complete the project in 46 weeks, up to 440 hr per week (rather than the estimated 350 hr per week) would have to be spent.

As it turned out, the project used approximately 1600 hr more than initially estimated and maximum weekly resources were slightly more than original estimates (371 hr/week instead of 350 hr/week). If these corrected figures for Ka and Yd are used in the analysis, then Ta, the date for acceptance testing, is 60 weeks instead of the actual 62 weeks—an error of only 3%.

OVERHEAD

Overhead is often an elusive item to pin down. In our projects three aspects of development have been identified: programmer effort, project management, and support (librarians, typing, etc). In one project (Fig. 3), programmers accounted for about 80% of total expenditures with the support activities taking about one third of the remainder. In addition, only about 60% of all programmer time was accountable to explicit components of the system (as reported on the Component Status Report). The remaining time includes activites like meeting, travel, training sessions, and other activities not directly accountable. This "loss" of time is a signficant overhead item which must be considered in developing accurate project budgets.

ERROR ANALYSIS

The correction of errors in a system is the major task of integration testing. Even a simple counting of errors can be useful as a management estimating tool. Figure 4(a) represents the number of error reports reported per week on one NASA project. It remained surprisingly constant over the testing stage. However, the more interesting measure is the *handling rate*[7], or the number of different components altered each week (Fig. 4b).

Consider the following set of assumptions:

1. The number of errors in a system is finite, but unknown.
2. The probability of finding an error is proportional to the number of individuals working on the problem.
3. The probability of finding an error is random and uniformly distributed.

These three assumptions lead to a Poisson distribution

$$y = e^{-at}$$

as the probability of an error remaining after time t. Furthermore, if we include the assumption that the probability of fixing a found error (as opposed to creating a new error by fixing the previous error) is the function $a = bt$ (e.g. errors are "easier" to find as you get "good at it"), then the resulting distribution is the same Rayleigh curve described previously[5].

Therefore, if N is the total number of errors in a system, and if h is a measure of the maximum number of errors found per week, then the number of errors found per week agrees with the curve:

$$y = 2Nht\exp(-ht^2).$$

A preliminary evaluation of the data of Fig. 4 (and other projects) seems to bear out these assumptions. Therefore, by using least squares techniques, the following algorithm can be used to measure testing progress:

1. Collect data on errors reported for several weeks.
2. Use least squares to fix a curve to this data. This gives a measure of N (modules handled) and h (a measure of maximum errors found).
3. N gives the number of modules in error in the system, however, this value can never be reached

INITIAL ESTIMATES FROM GENERAL PROJECT SUMMARY	
Ka, Resources needed (hours)	14,213
Ta, Time to completion (weeks)	46
Yd, Maximum resources/week (hrs)	350
COMPLETION ESTIMATES USING RAYLEIGH CURVE	
K, Resources needed (hours)	16,151
Estimated Yd with Ta fixed (hrs)	440
Estimated Ta with Yd fixed (hrs)	58
ACTUAL PROJECT DATA	
K, Resources needed (hrs)	17,742
Yd, Maximum resources (hrs)	371
Ta, Completion time (weeks)	62
Ta, estimated using actual values of K and Yd (weeks)	60

Fig. 2. Estimating Ta and Yd from General Project Summary data.

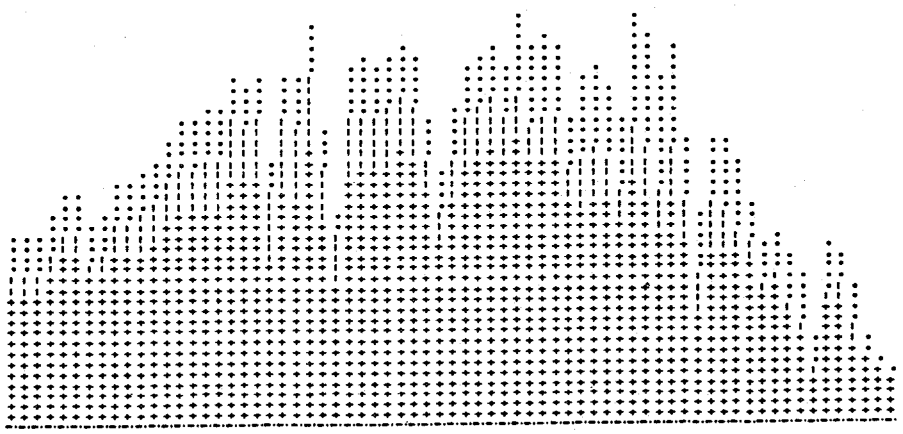
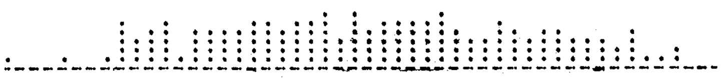
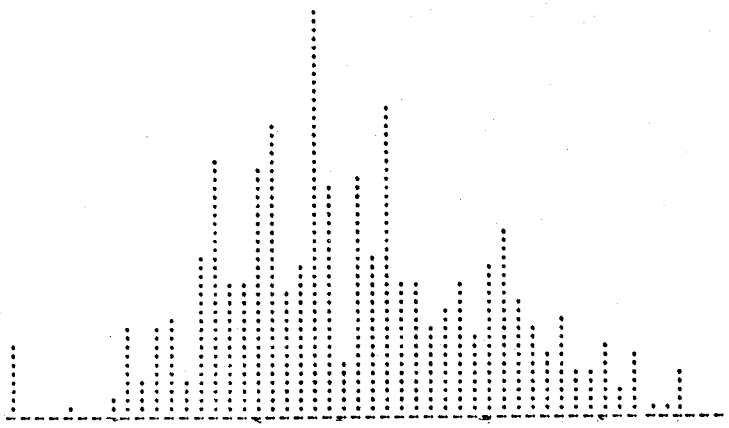


Fig. 3. Resources expended on various developmental activites. +, Programmer effort. −, Management effort. ·, Support effort (librarians, typing, clerical, etc.).

(a)





(b)

Fig. 4. Handling and report rates on one project. (a) Report rate by week. (b) Handling rate by week.

exactly. Compute the time needed to get the number of remaining errors to an "acceptable" level[8].

The project represented by Fig. 4 shows the practicality of this measure. This project has a total of 1115 components that were handled. A least squares fit yielded an N of 1024.9 and an h of .0009024 with a correlation of .7264. This figure of 1024 was only an error of 8% in the true handling rate. Current research is studying this aspect of errors in order to refine this measure further.

Acknowledgements—We would like to acknowledge the contributions and cooperation of Mr. Frank McGarry, head of the Systems Development Section of NASA Goddard Space Flight Center. He has been instrumental in organizing the Laboratory and in interfacing with the contractor in order to see that the data is collected reliably and timely. We would also like to thank Computer Sciences Corporation for their patience during form development and their contributions to the organization and operation of the Laboratory.

REFERENCES

1. B. Boehm, Software and its impact: a quantitative assessment *Datamation* 97–103 (July 1977).
2. V. Basili and A. J. Turner, Iterative enhancement: a practical technique for software development. *IEEE Transactions Software Engng* **1**(4), 390–396 (1975).
3. D. Teichroew and E. A. Hershey, PSL/PSA: a computer aided technique for structured documentation and analysis of information processing systems. *IEEE Transactions Software Engng* **3**(1), 41–48 (1977).
4. R. Wolverton, The cost of developing large scale software. *IEEE Transactions Comput.* **23**(6), 615–636 (June 1974).
5. P. Norden, Use tools for project management. *Management of Production.* (Edited by M. K. Starr) pp. 71–101. Penguin Books, Baltimore, Maryland (1970).
6. L. Putnam, A macro-estimating methodology for software development. *IEEE Computer Society Compcon*, pp. 138–143, Washington, D.C. (Sept. 1976).
7. L. A. Belady and M. M. Lehman, A model of large program development. *IBM Systems J.* **15**(3), 225–252 (1976).
8. J. D. Musa, A theory of software reliablility and its application. *IEEE Transactions Software Engng* **1**(3), 312–327 (Sept. 1975).