(12) LEVEL II

AD A091923

School of

Information and Computer Science

GEORGIA INSTITUTE OF TECHNOLOGY

12

14 GIT-ICS-80/09

SOFTWARE PROJECT FORECASTING†

10 RICHARD A. DeMILLO*
RICHARD J. LIPTON**

12 79

11 October 1980

*School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

**Department of Electrical Engineering and Computer Science
Princeton University
Princeton, N.J. 08750

†Work supported in part by Office of Naval Research, Grant 15 N00014-79-C-0231

410044

JOB

SOFTWARE PROJECT FORECASTING

Richard A. DeMillo
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

Richard J. Lipton
Department of Electrical Engineering and Computer Science
Princeton University
Princeton, NJ 08750

INTRODUCTION

A characterizing feature of the sciences is that they seek to explain, describe and predict phenomena. While there are varying degrees of exactness in the sciences (the predictions of physics are of a quantitative character, different than, say, the predictions of economics) the basis of scientific activity is rational and objective. This is what distinguishes *economics from astrology* -- even though it might be argued that economic and astrological predictions are equally vague, economic predictions are the result of rational analysis of evidence. Thus we intend to divide sciences *from non-sciences on the basis of the rational nature of the activities*; for a detailed discussion, see [1].

Let's now look at the problem of "measuring" software. It is evident from the remaining papers in this collection that aside from a well-founded concern over methodological issues, the principle aim of studying software metrics is not the static determination of software properties, but is rather *the scientific prediction of phenomena during the software lifecycle*. Indeed, Perlis, Sayward and Shaw point out (cf. [2]): "The purpose of software metrics is to provide aids for making the optimal choice... at several points in the life cycle." They go on to illustrate the nature of

the decision points.

- How long will it take to produce the software?
- When will it have to be replaced?
- What are the manpower requirements?
- How will the availability of tool X affect factor Y?
- How close to its resource limits will the system run?
- Will it run reliably?

These are manifestly problems of prediction. As intriguing as the issues of explanation and description may be -- our ability to usefully model software is still primitive -- the exigencies of governmental and commercial computing demand a reasonable facility in forecasting critical software project parameters. We will address this forecasting problem in the sequel. We pretend no well-formed answers here. In fact, our goal is the rather modest one of pointing out that there is a scientific (although possibly inexact) component of the problem that is not adequately conveyed by the term "software metrics": the use of past information to predict the future of a software system.

ANALOGIES

Inevitably, primitive sciences are compared to physics, the reliable standard of scientific rigor and success. Of course, inexact sciences fare very badly in the comparison, but often the reason they fare so badly is not understood. Physics stands almost alone among the sciences in the exactness and simplicity of its theories. The price paid is the complexity of the situations that can be profitably handled by the theories. Predicting the behavior of complex systems -- particularly those involving human

interactions -- is almost never carried out by deducing from first principles, that is, from physical theory. Tim Standish [3] points out that most scientific knowledge is organized so that phenomena at one level can be explained in terms of (or "reduced to") phenomena at a more basic level; for example, physical chemistry explains chemical behavior in purely physical terms. Only rarely, however, is it possible to compose several such reductions in intellectually manageable fashion. Thus while it is possible to imagine physical explanations of biological phenomena, biological explanations of psychological phenomena, and a psychological basis for social behavior, it is extremely unlikely that there will ever be a physical theory of social behavior.

Jim Browne is correct: "There are analogies from other sciences."[4]. We even agree that the fundamental issues are predictive and phenomenological. We differ in the choice of analogies, and we also differ on the role that measurement plays in constructing useful forecasts. It seems very unlikely that the theory of software forecasting is like physical theory at an early stage in its development. We have argued elsewhere [5] that software exhibits much complexity and ad-hocery, features that cannot easily be abstracted from real situations or simplified with approximations. The prediction problem for software is more akin to the corresponding problems in those disciplines that deal with complex systems; rather than turn to physics for our methodology, we should turn to the less exact sciences -- fields such as meteorology, economics, political science and even the construction industry. These are fields in which the explanatory component is subjugated to the predictive component primarily because of the extreme public and social significance of the predictions.

Why distinguish at all between explanatory and predictive theories? After all, predictive and explanatory assertions are logically equivalent. They both use evidence to convince a listener of an hypothesis. They may both give "laws" concerning an effect X; the exactness of the laws may vary from that of the quasi-laws [1] (X is asserted to be present except in certain exceptional cases) to the more exact statistical laws (X is asserted to be present in a stated fraction of the observations) to the mathematically exact physical laws. Predictions and explanations are, however, distinguished in a fundamental way: in explanatory theories the hypotheses concern events which are past, while predictions are hypotheses concerning future events. In logical terms, for an explanation to establish its conclusion, it must be more credible than its negation. On the other hand, a prediction must only be rendered more tenable than the alternatives!

METEOROLOGY

The physical basis of meteorological theory fits comfortably on a moderately large blackboard. It consists primarily of six equations of fluid dynamics which locally predict the state of the atmosphere from the current state:

STATE(new) = f(STATE(old)).

By observing pressure, temperature and other meteorologically interesting variables (in fact, only six independent variables and a few thermodynamical constants are involved) and calculating their rate of change, the state equations allow extrapolation over short time periods to new, or predicted, values (see the accounts in [6] or [7]).

It is amusing that the first attempts at a meteorological theory were directed toward an explanatory theory of the weather; this was attempted by ancient Greek philosophers. But predicting the weather in the vicinity of the Mediterranean Sea is not a very pressing concern. To Europeans a thousand years hence, however, the state of the weather was a subject of intense interest -- crops, harvests of fish, and trade routes depended on the vagaries of the less temperate climate. It is in the transition from Aristotle's speculation concerning the nature of the winds to the modern large-scale calculations that give rise to daily forecasts that there is a lesson for software forecasting.

The first stage in predicting weather most likely took the form of attempting to codify the "portents" of change: "They....were not so much concerned with explaining why weather happened as they were in predicting it, and [the early Europeans] gradually built up a huge folklore and literature of portents -- the unseasonable migration of birds, hibernation of wild beasts, unusual sexual behavior of farm animals, color of the sunset,..." ([6], p.128). Such understanding is entirely qualitative, and although it is tempting to try, building a quantitative, predictive theory by improving on the portents is not productive. What was needed in meteorology was, first, a proper concept of primary data. Not until the middle of the seventeenth century were there means to observe atmospheric temperature and pressure. Measurements by themselves told little about climate beyond what was obviously revealed directly by the observation (e.g., "It's cold out there!") It was after Newton (and later Bernoulli, Euler and Boyle) that a coherent fluid mechanics began to emerge.

It took nearly two hundred years to get from Bernoulli to the six equations of state for the atmosphere. These equations are, however, analytically intractable. The only hope of obtaining high quality quantitative predictions of the weather lay in massive computation. A proposal made by Richardson in 1922 involved the hand calculation of such nonlinear systems of equations by numerical techniques that are close to what is used today [7]. Without computers, however, Richardson could only speculate on the actual mechanism of carrying out the necessary calculations:

> [Richardson] describes a phantasmagorical vision of the "weather factory" -- a huge organization of specialized human computers, housed in something like Albert Hall, directed by a mathematical conductor perched on a raised pulpit, and communicating by telegraph, flashing colored lights, and pneumatic conveyer tubes...In this fantasy, he estimated that, even using the newfangled desk calculators, it would take about 64,000 human automata just to predict the weather as fast as it actually happens in nature. Richardson's preface concludes with a rather wistful but prophetic statement: 'Perhaps someday in the dim future it will be possible to advance the computations faster than the weather advances and at a cost less than the saving to mankind due to the information gained. But that is a dream' ([6], p.138).

The nearly simultaneous advance of sophisticated numerical analytical techniques and high speed digital computation allowed the fulfillment of Richardson's dream in essentially its original form.

The characterizing features of modern meteorological forecasting are that, first, extensive primary data are gathered, second, accurate microscopic theories of atmospheric behavior are available, and, third the microscopic prediction -- obtained from the local data -- are pieced together using massive computation.

A DIGRESSION ON MEASUREMENT

Since measurement of atmospheric pressure and temperature enter into our meteorological analogy, we should digress for a moment to consider the notion of measurement of software. The remaining papers in this collection refer to software "metrics". The term metrics refers to "indices of merit that can support quantitative comparisons and evaluations..." [8]. In the context of predictive modelling (that is of predicting the future from the past), it is more convenient to think in terms of observable software properties -- particularly those that can be numerically characterized and objectively recorded -- in other words, to think in terms of measurements instead of metrics. The distinction is not totally pedantic. There is a rich theory of measurement that guides the development of other models [9,10,11,12], and most importantly can be used to insure that there is a precise sense in which hypotheses that are formulated about the measured quantities are meaningful.

By a measurement is meant the assignment of numbers to represent properties of material systems; since by a system we mean a collection of objects or events, the properties of the system are given by relations between the objects/events. For reasons of intellectual economy a scientist usually isolates one aspect of the system to study; that is, he focuses on one relational system. So, a measurement -- an assignment of numerals to objects or events according to certain rules [9] -- can be defined to be a mapping f from a relational system (A,R), where A is a set and R is a binary relation defined on A, to a set of numbers. Since the numbers "represent" the relation, we should insist that:

$$aRb \quad \text{iff} \quad f(a) > f(b),$$

whenever a and b are objects in A. More concisely, measurements are defined to be homomorphisms that preserve certain basic relations.

Thus, a basic measurement of, say, temperature is obtained by the assignment of a number by a well-defined rule (e.g., the height of fluid in a standard thermometer). This homomorphism is not uniquely defined, however. It is possible -- and common -- to define differing scales for measuring temperature. The scientifically meaningful statements that can be made about temperature do not depend on the scale; that is, they remain valid under rescaling. Just which rescalings are allowed depends on the properties of the relational system (A,R). Similarly, scale types can be characterized by the admissible rescalings as summarized in Figure 1 (see [10]).

ADMISSIBLE TRANSFORMATIONS	SCALE TYPE	EXAMPLE
$\Phi(x) = x$	absolute	census counting
$\Phi(x) = ax$ $a>0$	ratio	time interval length
$\Phi(x) = ax+b$ $a>0$	interval	time, temperature
$x \geq y$ implies $\Phi(x) \geq \Phi(y)$	ordinal	preference
Φ is 1-1	nominal	labels

Figure 1. Common Rescalings

Strictly speaking, an empirical statement is meaningful provided its truth is invariant under an admissible transformation. We can safely assert that 100 degrees C is the boiling point of water, since the statement

is true under the rescaling a = 9/5 and b = 32. It makes no sense to assert that the temperature on March 15 is twice the temperature on November 4, since temperature is not defined on a ratio scale -- the ratio of temperatures depends on the scale and is therefore not invariant under the rescaling.

As a warm-up, let's look at some empirical statements about software.

I. The length of Program A is at least 100.

II. Program A is 100 lines long.

III. Program B took 3 months to write.

IV. Program C is twice as long as Program D.

V. Program C is 50 lines longer than Program D.

VI. The cost of maintaining Program E is twice that of maintaining Program F.

VII. Program F is twice as maintainable as Program E.

Statement I does not make reference to a particular scale, so it does not make sense, whereas Statement II does make sense. Similarly, Statement III is a perfectly reasonable factual statement. If the expected scales are provided for Statements IV and VI, they are meaningful, but as written they are technically meaningless. Statement V, however, refers to a ratio scale, on which intervals make sense. Finally Statement VII forms a ratio on an ordinal scale, which is meaningless.

From the standpoint of measurement theory, many of the derived measurements of software that have been proposed [13] are meaningless.

Example 1. Programming Effort Equation

The total effort E in number of man months is

$$E = 2.7v + 121w + 26x + 12y + 22z - 497.$$

The interpretation of the variables and their scale types are given in the following table.

VARIABLE	INTERPRETATION	SCALE
v	number of instructions	ratio
w	subjective complexity	ordinal
x	no. external documents	absolute
y	no. internal documents	absolute
z	size in words	ratio

This example illustrates a common shortcoming of current attempts at fundamental and derived measurements. Not only is the equation dimensionally inconsistent (no. of documents + no. of instructions + words + complexity ≠ man-months), it does not rescale: the truth of the equation cannot be invariant under the required transformations.

Example 2. Another Programming Effort Equation

The total effort E in man-months is

$$E = 5.2(L^{**}.91).$$

In this equation, L is the program size in thousands of lines of code, so that both E and L are expressed in a ratio scale. But the measurement is not invariant under the transformation L --> aL' and so is meaningless.

Example 3. Life Cycle Cost

A basic measurement that does satisfy the requirements of measurement theory is the equation for life cycle cost in dollars LC, expressed as a function of the cost in man years, M,
and the average cost per man year, C:

$$L = MC.$$

Example 4. Error Seeding

The technique of introducing artificial errors into a program, testing the program and determining the ratio of seeded to natural errors can be used to estimate the number of initial errors in a program, by the equation

$$N = ST/C,$$

where N is the estimate of the initial number of errors, S is the total number of errors sampled, T is the number of "tagged" or seeded errors and C is the number of errors in common in the counts S, T. Since the only admissible transformation is the identity, the equation is technically meaningful.

As a guide to fundamental measurement in software forecasting, measurement theory suggests a more thorough study of the underlying relations to be measured. In particular, if the underlying mechanisms are to be exposed, the most basic methodological analyses suggest that it is prudent to at least determine the scale type first. At least that leads the investigator to propose and experiment on meaningful hypotheses.

REALISTIC FORECASTING GOALS

Examples 1 and 2 of the previous section illustrate a good deal of what is wrong with current approaches to software metrics. Not only do the equations suffer from the technical defects cited above, they are also unlikely candidates for useful laws: they are too simple! The number and quality of interactions that must take place to produce a software system mitigate against a forecasting problem that can be easily solved on a hand calculator. The forecasting models that are the most realistic are also the most demanding in terms of computation and data gathering. For example, the controversial world dynamics model of Forrester [14] requires well over 500 pieces of primary data and massive computations.

By analogy to the meteorological prediction problem, we, therefore, reject the idea that there can be a single "correct" measurement of software. Instead, we look for many single measurements to aggregate - the large number of factors involved mitigate against the simplistic models cited above; in fact, they necessarily imply that the prediction problem must be associated with a computer-based solution! That is, we look for simple microscopic laws which interact in macro effects which are -- either by necessity or by our lack of knowledge -- beyond human understanding and by (possibly massive) computation combine them to obtain a prediction.

There are two relevant approaches to forecasting that deserve our attention here. The first approach is the classical econometric time-series approach to forecasting [15]. In this approach one looks for statistically meaningful patterns in past data and uses these patterns to predict future patterns. It seems to us that this approach is well-

developed and has been applied with some success to certain kinds of software lifecycle modelling [16].

From the standpoint of basic research, however, the traditional forecasting approach is not very satisfying, since it is an admission that the underlying mechanisms have not been understood. Returning to the meteorological analogy, it is an attempt to extend the portents. But that seems to be the stage of our current understanding of the software lifecycle.

The exact approach to forecasting seems to require many more insights into the various facets of the software lifecycle than we currently have at our disposal. Rather than wait for the software equivalent of Newton (or Galileo, for those who believe that we cannot even measure temperature), we might try to use large-scale computation to build upon the primary data that we can collect. It should be possible to partition a wide variety of programming tasks into discrete, classifiable subtasks that are repeated anew for each project. We can imagine, for example, a catalog of subtasks (such as terminal handlers, hash table routines, and report writers) which are common in various applications software. Notice that we do not claim that these are "off-the-shelf" components -- we merely claim that they must be recreated in approximately the same form for each new job. This catalog will be quite extensive, but it will be conceptually simple to structure and use.

The principle data gathering activity is to determine the cost estimates for each of these subtasks. These costs are influenced by many factors, including the potential application, the skill and experience of the programmers and the restrictions imposed by the programming environment.

There are many sources for such estimates. First, there is a great deal of historical data which can be carried forward; after all we do have considerable experience with software projects and this experience can be codified. Second, we have expert advice concerning the cost of the projects. Third, experimentation can be carried out. Fourth, the cost estimation is from managed projects so feedback can be used to correct prior estimates.

If the data on the primitive tasks that make up the software system is reliable, then the task is to "piece together" a forecast of the total system cost by large-scale computation. By "reliable" we mean that the measurements have an accurate mean and small standard deviation, since in that case the Central Limit Theorem [17] guarantees that the overall estimate will have a small error term (in fact one that grows as the square root of the total number of terms). It is important to distinguish in this approach between using standard cost and estimation data and advocating the use of "standardized software components". Perhaps an analogy to a more familiar cost forecasting problem will make the point more clearly. To estimate the cost of a house, a contractor will consult extensive data sheets on the cost of installing doors (how big?, how many?) and the hundreds of other basic components of a dwelling. These are not prefabricated items, they must be constructed completely from basic specifications and customized for the task at hand, but they are enough alike to permit an accurate assessment of the expected cost of construction. A cost estimate from a builder is pieced together from such estimates.

Now, it is entirely possible that this approach requires inordinate overhead; but there is still room for applying computational power to chip away at the forecasting problem from historical data. Often, the scale

which one uses to assess the software project is even weaker than an ordinal scale: often all that is required is a measure of cohesion between software projects. A manager may only need to know whether the current project is enough like apparently similar projects which have succeeded (or failed) to justify his decision. In this situation the computation needed is a similarity analysis of the important project factors. A large clustering analysis of projected software tasks and historical data may provide such information [10]. The principle task in such an endeavor is to isolate the important factors through data gathering and experimentation.

SUMMARY

We have argued that a major use of software metrics is in the forecasting problem for software projects. By analogy with weather forecasting, we may characterize the current state of knowlege in software forecasting as the gathering of "portents." While these may be useful and sometimes decisive in project management, they are prescientific and qualitative. Further, it seems very unlikely that the portents can be developed into a useful theory of forecasting. To develop scientific forecasting tools, a rational way of predicting the future from historical primary data is required. It is also important that the primary data and the measurements used to obtain it satisfy some basic methodological requirements -- for example, the hypotheses developed from the measurements should be meaningful in the sense implied by measurement theory. → next page

Among the rigorous approaches to the prediction problem we distinguish the statistical and the exact approaches. We specifically reject the notion that such complex phenomena as software lifecycles can be dealt with

in a global way using computationally simple "laws". The statistical approach, seeking to predict future events on the basis of historical patterns, seems to be an attractive short range approach to the forecasting problem. There is certainly an extensive body of theory from econometrics and related areas which can be brought to bear on software forecasting. Unfortunately, the statistical approach is a recognition that the underlying mechanisms are not understood. We turn, therefore, to the exact approach. In the exact approach a great deal of effort is spent in attempting to understand -- or to at least quantitatively assess -- the microscopic prediction problem. The goal of the exact method is to be able to apply largescale computation to many micropredictions to synthesize a quantitative forecast. There may even be useful aggregations of statistical and exact techniques which give forecasting models. In both approaches, data gathering is an essential activity; it is, therefore, important to settle on the fundamental measurements to be performed on the software.

REFERENCES

[1] Olaf Helmer and Nicholas Rescher, "On the Epistemology of the Inexact Sciences," Rand Corporation Report No. R-353, February, 1960.

[2] Alan Perlis, Fred Sayward, and Mary Shaw, Unpublished notes on software metrics, April, 1980.

[3] Tim Standish, Notes on Software Metric and ADA, January 1980, Las Vegas, Nevada meeting of ONR Software Metrics Group.

[4] Jim Browne, "A Philosophy and Justification for Empirical Software Engineering and Software Science," Unpublished Notes 9/13/79.

[5] R. DeMillo, R. Lipton and A. Perlis, "Social Processes and Proofs of Theorems and Programs," Communications of the ACM, May, 1979, pp. 271-280.

[6] Philip Thompson, "The Mathematics of Meteorology," in Mathematics Today, edited by Lynn Steen, Springer-Verlag, 1979, pp. 127-152.

[7] Molly Gleiser, "The First Man to Compute the Weather," Datamation, June, 1980, pp. 180-184.

[8] Alan Perlis, Fred Sayward and Mary Shaw, unpublished Notes.

[9] Fred Roberts, Measurement Theory, Addison-Wesley, 1979.

[10] Michael Anderberg, Cluster Analysis for Applications, Academic Press, 1973.

[11] Norbert Weiner, "A New Theory of Measurement: A Study in the Logic of Mathematics," Proceedings London Math. Society, 1919, pp. 181-205.

[12] D. Krantz, et al., Foundations of Measurement, Vol. 1, Academic Press, 1971.

[13] Data and Analysis Center for Software, "Quantitative Software Models, March , 1979.

[14] J. Forrester, World Dynamics, 2nd Edition, MIT Press.

[15] Robert Pindyck, Econometric Models and Economic Forecasts, Mc Graw-Hill, 1976.

[16] L. Putnam and A. Fitzsimmons, "Estimating Software Costs," Datamation, September, October and November, 1979.

[17] W. Feller, An Introduction to Probability Theory and its Applications, Vol. I, Wiley, 1968.