Estimation of Growth Parameters for a Software Development

Chandrakanth G Pujari
Research Scholar Sathyabhama University
Chennai, Asst. Prof., MCA Dept. Dr.AIT Bangalore

Dr. Seetharam K
Professor, Dept. of CSE, Bangalore

ABSTRACT
Software reliability growth models started to be developed in an era where the waterfall model was king (or queen), but they are less useful in modern approaches to software development. Thus, we have either to invent completely new ways of capturing the information that is hidden in failure data or we have to adapt the usage of the software reliability growth models to current ways of developing software. This paper is to show how statistical techniques can be used to manage the software development process, be it for productivity assessment or for source selection when software productivity data can be indexed, as in a time series, then growth-curve models can be used to track the data for trends, and for making projections. There is a vast amount of literature on growth-curve models and consequently the choice of models is large. However, for purposes of illustration, we selected a simple power rule model, and motivated its relevance for monitoring software productivity the chosen model when suitably transformed is a random coefficient autoregressive process which, we recall, is also one of the dynamic linear models used to describe software inter failure times.

Keywords
Software Reliability, Quality, Lognormal Distribution, Posterior Distribution, Productivity

1. INTRODUCTION
One problem with software productivity studies is the difficulty of gathering sufficient data to support a credible statistical analysis. It is desirable that programmers gather basic data on their personal performance on every program they produce. The data described in the following have been gathered in response to this theme. Regarding the productivity of large-scale tasks or team activities, there are problems with gathering data. First is that team projects take several months to complete, and so this would entail much time and effort vis-à-vis the data collection. Second, the stability of such data would depend on the composition of the team. Thus, for example if team member were to leave or new members were to be added during the project, the prior data would not be likely to be comparable to the new data. In table A column 2, we show the minutes taken by a highly experienced programmer to develop a line of code for 10 programs of 10 programming languages varying sizes taken from Caper Jones [1,2] development includes designing, implementation, and unit testing. The 10 programs in question were not similar in sizes, but the environment under which they were developed was, for all intents and purposes, identical; that is, it was carefully controlled. Thus conclusions and projections of productivity based on these data would be valid only for this programmer, working under the tightly controlled environment. The traditional way of predicting software reliability has since the 1970ies been the use of software reliability growth models [3,4,5]. They were developed in a time when software was developed using a waterfall process model. This is in line with the fact that most software reliability growth models require a substantial amount of failure data to get any trustworthy estimate of the reliability. Software reliability growth models are normally described in the form an equation with a number of parameters that need to be fitted to the failure data. A key problem is that the curve fitting often means that the parameters can only be estimated very late in testing and hence their industrial value for decision-making is limited. This is particularly the case when development is done, for example, using an incremental approach or other short turn around approaches. A sufficient amount of failure data is simply not available. The software reliability growth models have initially been developed for a quite different situation than today. Thus, it is not a surprise that they are not really fit for the challenges today unless the problems can be circumvented. This paper addresses some of the possibilities of addressing the problems with software reliability growth models by looking at ways of estimating parameters in software reliability growth models before entering integration or system testing. Humphrey (1991) [9] have analyzed these data using the methods of classical time series analysis. Their productivity projections, based on an exponential smoothing formula, were reasonable when compared with actual data. However, their approach relied only on past data to make future projections and did not take into consideration. a knowledge of learning theories and application environments. Further more, their approach did not provide any insights about growth (or decay) in productivity. Specifically, was this programmer still experiencing a learning phenomenon or did he or she reach a point of saturation whereby learning was be facto minimal? Can the techniques described here provide an answer to the preceding question? In what follows we explore this and related issues. In column 3 of table A we show the values of Y(t) = log X(t), t= 1,2,……..10 and we show a plot X(t) versus t. this plot shows that X(t) fluctuates quite a bit, alternating between an up and a down, but otherwise fails to reveal any underlying trend. A plot of Y(t) versus t as shown in graph 1and 2 respectively.

To apply the methodology of this section, we follow the guidelines for choosing hyper parameters and make the following choices; $\sigma 2$ (v) = 0.35, $\delta 1$ = 1, $\gamma 1$ = 5, m1=1,and s1 = 0.35. these choices reflect a strong commitment to the proposed model, and a strong a priori opinion that there is neither growth nor decay of productivity. the latter position is appropriate because even though the programmer is an

experienced one, columns 4,5 and 6 of table A show the values of mt, θ(t), Y+(t); these are obtained from the following equations respectively St ={St-1 – (St-1)2 * (Y(t-1))2}/P(t) , Y+(t+1) =Mt*Y(t), for t = 1,2,…….. Column 7 of table A compares the one-step-ahead predictions Y+(t) versus Y(t), t = 2,3,……..,10, via their absolute differences, and columns 8 and 9 gives the 68% prediction limits for the one-step-ahead projections. A plot of these prediction limits is shown in graph2; the plot indicates the extent to which the prediction limits cover the observed Y(t). since Y+(t+1) = Mt *Y(t) the predicted values tend to be relatively close to the most recent observed value; this is borne out by a comparison of the entries in column 3 and 6 of table A.

2. ANALYSIS

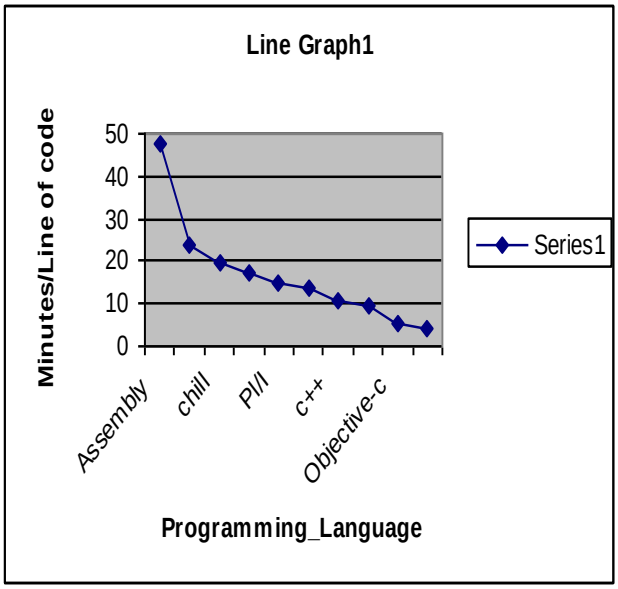
In column 3 of table we show that the values of Y(t) = logeX(t), t=1,…10, and in line graph, we show a plot of X(t) verses programming language. This plot shows that X(t) fluctuates quite a bit, alternating between an up and down, but otherwise fails to reveal any underlying trend. a plot of Y(t) verses programming language is shown in line graph2. In this graph a strong a priori opinion that there is neither growth nor decay of productivity. The later position is appropriate because even though the programmer is an experienced one , in column 4,5 and 6 of tableA show the values of mt , θ(t), and Y+(t) ; these are obtained via equations. Column 7 of tableA compares the one-step-ahead predictions Y+(t) verses Y(t), t=2,3,……,10, via their absolute differences, and columns 8 and 9 give the 68% prediction limits for the one-step-ahead projections. A plot of these predictions limits is shown in line graph2, the plot indicates the extent to which the prediction limits cover the observed Y(t). since Y+(t+1) = mt * y(t) the predicted values tend to be relatively close to the most recent observed value ; this is born out by a comparison of the entries in column 3 and 6 of table A. In line graph3, we show a plot of mt verses programming language t , for t=1,2,….10. since the values of Mt tend to remain above 1 , for most values of t , there does not appear to be present any evidence of productivity growth. In line graph4 , shows a plot of θ+(t) versus t , for t=2,…..10. The uneven nature of this plot suggests there is no steady pattern of growth or decay from program to program. However , the number of times the plotted values exceed one, and the magnitudes of these variations, indicate a slight decay in productivity. This is also suggested by the plot of line graph2. In this analysis negative learning trend might be caused by the fact that the 10 programs from the Caper Jones textbook have a problem sequence of progressively increasing difficulty.

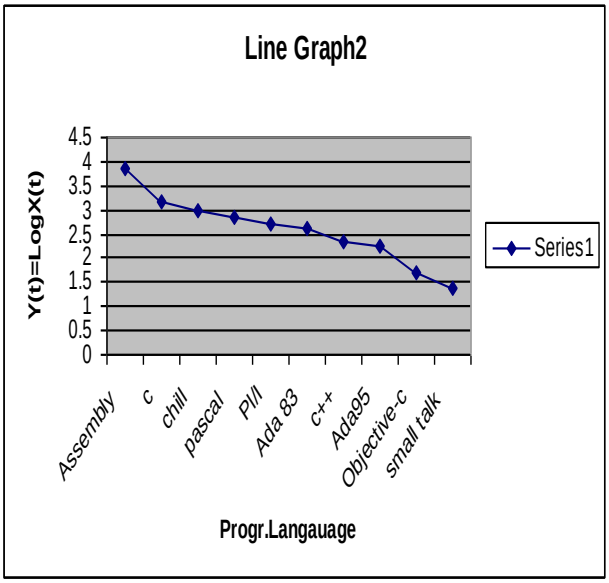
3. INFERENCE AND PREDICTION UNDER THE GROWTH CURVE MODEL

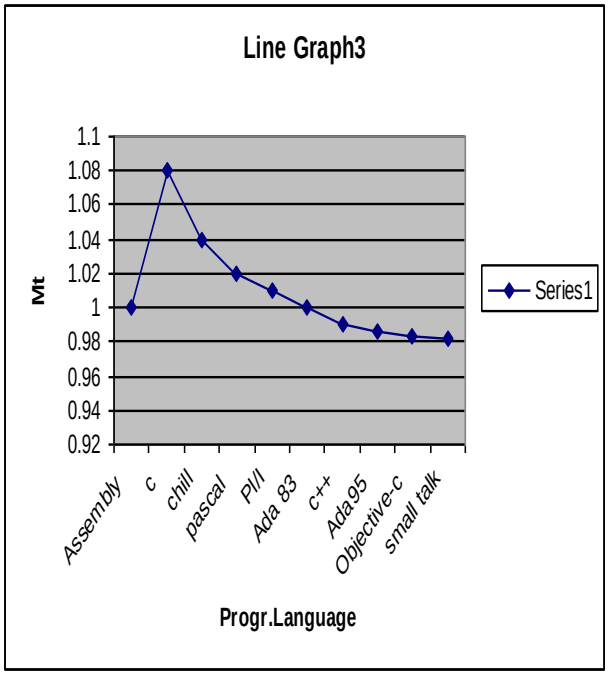
Let y(t) denote the observed value of Y(t), t=1,2,……, and suppose that y(t) = (y(1),…….y(t)) have been observed. Let mt denote the mean of the posterior distribution of λ given y(t); that is, mt is the updated value of m1 in the light of y(t). since mt conveys information about an overall growth or decay in productivity, a plot of mt versus t, for t=2,3,….., would suggest a steady growth in productivity if the values of mt were to lie below one ; otherwise, there is evidence of a decay in productivity[8,9].The determination of mt is relatively straight forward; specifically , with m1,s1, and σ2(v) specified, it can be shown below Mt = Mt-1 +St-1 * Y(t-1) * (Y(t)-Y(t-1)*Mt-1) / P(t) assessment about productivity growth or decay, from program to program, are provided by the parameter θ(t), t = 1,2,…….10 θ+(t) denote the mean of the posterior distribution of θ(t),given Y(t). note that Y+(t+1) denotes the predicted value of Y(t+1), given Y(t) = (Y(1),…….Y(t)), then Y+(t+1) = Mt*Y(t), for t=1,2,…….10 note that Y+(t+1) is a projection of productivity. It is needed to estimate the time and effort required to complete future tasks involving design, implementation, and unit testing. Associated with such estimates are measures of uncertainty. The upper (lower) projection limit for Y+(t+1), with a coverage probability of approximately 68% , is given by the formula **(Mt * Y(t) + √Wt) * [Mt * Y(t)- √Wt]** , where **Wt=δt[1 + (Y(t))2 (St + σ2(v))] / (Γt-2) Γt, δt** are the updated values of Γt, δt , respectively. In the light of y(t) . they go to determine the degrees of freedom parameter of the students t-distribution that is used to obtain wt. they are given as **Γt = Γt-1 + 1**,

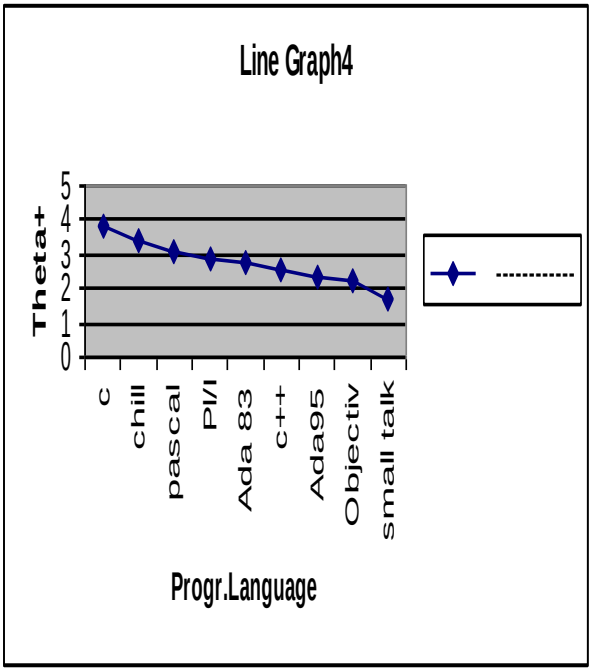
4. PRODUCING QUALITY SOFTWARE

We address two topics that are of interest to managers of software development Teams[6,7]. The first pertains to producing software within reasonable cost and time constraints. The second pertains to ensuring that the software produced is of acceptable quality. Statistical techniques have role to play here, and the aim of this paper is to highlight this role. When developing software, an issue faced by a manager is the prediction of development time and effort. A common approach is to first estimate the size of the program; say the number of lines of code, and then use some guidelines on productivity rates to arrive at the time and effort needed to complete the project.This paper focus on the use of historical data to estimate productivity rates. Good estimates of the number of lines of code are essential, and a plausible approach is to elicit specialized knowledge and then the use of the techniques of subjective priors based on elicitation of specialist knowledge to asses the required quantity. The introduction of a structured statistical framework for collecting and analyzing such data should be an important over current practice; it is explained in a growth-curve model for estimating software productivity. software productivity can be assessed by what are known as growth-curve models. Such models have been used for describing human learning experienced and have provided suitable fits to data on performance based activities. A characteristic feature of learning experience data is that the successive measurements tend to be auto correlated (i.e, related to each other). When developing software code, there appears to be the presence of an underlying learning process that has the tendency to improve the successive development activities.


Line Graph1
Minutes/Line of code
50
40
30
20
10
0
Assembly
chill
Pl/I
c++
Objective-c
Series1
Programming_Language


Line Graph2
Y(t)=LogX(t)
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Assembly
c
chill
pascal
Pl/I
Ada 83
c++
Ada95
Objective-c
small talk
Series1
Progr.Langauage


Line Graph3
Mt
1.1
1.08
1.06
1.04
1.02
1
0.98
0.96
0.94
0.92
Assembly
c
chill
pascal
Pl/I
Ada 83
c++
Ada95
Objective-c
small talk
Series1
Progr.Language


Line Graph4
Theta+
5
4
3
2
1
0
c
chill
pascal
Pl/I
Ada 83
c++
Ada95
Objectiv
small talk
Progr.Language

Prog.Lang t	Size in LOC	Min/LOC :X(t) Obs	Y(t):log X(t)	m_t	Estimation of growth parameter $\Theta^+(t)$	Predicted values $Y^+(t)$	Abs(Y(t)-$Y^+(t)$)	68% prediction intervals Low	UP
Assembly	375000	47.35	3.86	1	_____	_____	_____	__	_
C	190500	24.05	3.18	1.08	0.84	3.85	0.67	2.16	2.62
Chill	157500	19.89	2.99	1.04	0.97	3.43	0.44	2.03	2.33
Pascal	137500	17.36	2.85	1.02	0.969	3.11	0.26	1.94	2.11
P/L	120000	15.15	2.72	1.01	0.968	2.91	0.19	1.85	1.98
Ada 83	106500	13.45	2.6	1	0.968	2.75	0.15	1.77	1.87
C++	82500	10.42	2.34	0.99	0.95	2.6	0.26	1.59	1.77
Ada 95	73500	9.28	2.23	0.987	0.964	2.32	0.09	1.52	1.58
Objective C	43500	5.49	1.71	0.983	0.85	2.2	0.49	1.16	1.50
Small talk	31500	3.98	1.38	0.982	0.89	1.68	0.3	0.94	1.14

Table 1: Minutes per line of code, their projected values , and estimates of growth parameters for a specific programming languages

5. CONCLUSION

- In this paper a particular measure of productivity, approach is general and can be used to study the behavior of any data that evolve over time and are suspected to have growth or decay characteristics.
- The main virtue of our approach is its ability to assess underlying trends in the presence of wild fluctuations as shown in above graphs, and its ability to assess stage-by-stage growth or decay.
- With highly trended and relatively stable data, this approach should lead to superior predictions over the standard time series approaches, both because of its underlying structure, and because it is adaptive; that is, the model parameters are constantly updated in the light of new data.
- In line graph4, the uneven nature of this plot suggests there is no steady pattern of growth or decay from program to program. However, the number of times the plotted values exceed one, and the magnitudes of these variations indicate a slight decay in productivity this is also suggested by the plot of line graph2.
- In this paper slight negative learning trend might be caused by the fact that the 10 programs from the Caper Jones have a problem (and this program) sequence of progressively increasing difficulty

6. REFERANCES

[1] Capers Jones , "Estimating Software Costs: Bringing Realism to Estimating 2007", Tata McGraw-Hill ,Second Edition.

[2] Capers Jones , "Applied Software Measurement Global Analysis of Productivity and Quality 2008", Tata McGraw-Hill, Third Edition

[3] Andy Ozment, "Software Security Growth Modeling: Examining Vulnerabilities with Reliability Growth Models",Proceedings of the first workshop on Quality of Protection , Milan, Italy: Sept. 15th , 2005 .

[4] M. Xie, G.Y.Hong and C.Wohlin, "Software Reliability Prediction Incorporating Information from a similar project", Journal of Software and Systems, Vol.49, No.1, pp, 43-48, 1999.

[5] Alaa Sheta, "Parameter Estimation of Software Reliability Growth Models by Particle Swarm Optimization", AIML Journal, Volume (7), Issue(1), June,2007.

[6] M.Xie,G.Y.Hong, C.Wohlin, "A Practical Method for the Estimation of Software Reliability Growth in the Early Stage of Testing", IEEE 1997.

[7] A.L.Goel and K.Okumoto, "Time-Dependent Error Detection Rate Model for Software and other performance measures" IEEE Trans.on Reliability, Vol. 28, No. 3, pp. 206-211,1979.

[8] Chin-Yu Huang,et al.. "Effort-Index-Based Software Reliability Growth Models and Performance Assessment"IEEE,2000.

[9] Humphery, et,al.., Predicting (1991) "Predicting (Individual) Software Productivity, IEEE Transaction Software Engineering, 17 2: 196-207.

[10] N.E.Fenton and M.Neil, "A Critique of Software Defect Prediction Models", IEEE Trans. On software Engineering. Vol.25,No.5,PP,675-689.