Chapter

11

Staffing Factors in Software Cost Estimation Models

by Chris F. Kemerer and Michael W. Patrick

Abstract

Many software cost estimation models have attempted to incorporate staffing variables such as personnel experience and capability. This chapter surveys such models and evaluates the extent to which accounting for staffing variables can improve their software project cost estimates. Current measures of ability, such as years of experience, appear to be poor indicators of individual capability differences and do little to improve model accuracy. Suggestions for improving these models through expanded effort in theoretical development are proposed.

Introduction

Software projects have a well-deserved anecdotal reputation for being late and over budget. This anecdotal evidence is supported with a number of research studies. Jenkins, Naumann, and Wetherbe (1984) report that of 72 medium-scale (average 10 month) software projects in 23 major US corporations, only 9% were completed within the scheduled manpower effort. The average effort overrun was 36%. Most projects (48%) completed with 75% extra effort, but some projects required three, four, and even five times the manpower effort of the original estimate.

A University of Arizona survey found an average, self-reported (191 of 827) overrun of 33%, and that only 16% of the respondents felt that software projects "rarely" or "never" experienced cost overruns (Phan, Vogel, and Nunamaker 1988). DeMarco and Lister (1987) report that in their survey of 500 software projects, 15% of all projects were so late they were canceled, and a full 25% of large (over 25 person-year) projects were canceled.

Heemstra and Kuster's (1991) survey of almost 600 European organizations revealed that over 80% of their software projects failed to meet their original time schedule. As expected, the larger the project, the more likely it was to slip. Whereas 71% of all projects completed within 10% of original budget, only 45% of the largest (>200 person-month) projects were within 10% of their original manpower estimate.

Most recently, van Genuchten (1991) presents the results of a number of studies that provide similar findings. When surveyed as to why software was late, van Genuchten found that the two most common answers were that personnel were not available (i.e., some other project was late) and "product related reasons" including "complexity of application underestimated."

Software cost estimation models attempt to more accurately estimate the effort required to develop a software project by using a mathematical formula of expected project inputs. The most frequently used input measure is the estimate of the size of the program in lines of code. The output of the model is usually an estimate of the effort in person-months.

However, use of these models in their current state of development is not a panacea for project management overruns. The Heemstra and Kuster (1991) study also demonstrated that software cost estimation models are not widely used or particularly effective when they are used. Only 14% of the projects surveyed used any kind of algorithmic cost estimation model at all. There was no significant difference in the magnitude of schedule slips between those projects that used a model and those that did not.

It is widely believed in practice that the single most critical factor to manage on a project is the staffing. Teams of the "the best" staff are widely believed to significantly outperform average teams. This belief is even sometimes raised to challenge claims for new tools or methodologies, arguing that the project was "stacked" with talented individuals, who were the primary reason for the success rather than the new tool. Given the belief in the importance of this factor, some cost estimation models attempt to improve their estimates by incorporating differences in the ability of the individual project team members.

This chapter surveys the published empirical literature that attempts to explicitly model the effect of alternative staffing policies. The main results of this survey are that, despite research spanning approximately two decades, little progress has been made in modeling the effect of these variables. Some suggestions are made for improving these models through greater incorporation of a stronger theoretical base.

Cost estimation models

The cost estimation modeling literature has been recently reviewed (Kemerer 1991) and will only be briefly summarized here. All software cost estimation models attempt to predict the effort as a function of various measured "factors":

Effort = <f>(factor1, factor2, ...)

Several researchers have measured a variety of factors and run a multiple regression to determine the most significant factors. For example, Walston and Felix

(1977) collected data on 68 factors, finding 29 to be of significance, and Chrysler (1978) collected data on 24 factors and found 14 to be statistically significant.

In all cases, the most significant variable was the actual "size" of the project, usually measured as lines of code. To improve accuracy, many models multiply the baseline effort by a modification factor $M(x_1, x_2, \ldots xn)$, which is a function of additional independent factors $x_1, x_2, \ldots xn$. The detailed COCOMO model, for example, uses 16 such modification factors (15 plus a "mode" overall variable). Many researchers feel that one of the most important modification factors is some measure of the differences between the individuals involved in the development team (Boehm 1981).

Table 11.1 summarizes the survey of software cost estimation model studies. Most of the columns are relatively self-explanatory, but the column marked "Productivity Output Measure" requires some further discussion. This column indicates how the study measured output, typically expressed in lines of code. This survey of software cost estimation models reveals a wide diversity in the method of counting lines of code. Most researchers did not count comments, but did count data (non-executable) statements. All studies counted new lines of code; Table 11.1 indicates whether reused (modified) code was counted as well. The columns indicate with a "Y" whether Executable, Data, Comment, and Overhead (developed but not ultimately used) lines of code were counted by the study.

The treatment of reused code varies significantly. Some studies, such as Walston and Felix (1977), ignore reused code completely. This was probably because it was not

TABLE 11.1 Surveyed studies.

Study	Num Proj	Min Size (KLOC)	Max Size (KLOC)	Avg Size (KLOC)	Type	Data Source	Language	Productivity Output Measure (Lines of Code)	Productivity Input Measure (Phases of Effort)
								N: New Lines R: Reused Lines E: Executable D: Data Lines C: Comments O: Overhead	R: Requirements Phase F: Functional Spec Phase D: Design, Code, Unit Test S: System Test Phase M: Management Effort P: Paperwork (Documentat'n) U: Units of Effort Measured
								N R E D C O	R F D T S M P U
[Sackman 68]	12	0.65 (Kobj)	6.14 (Kobj)		Experiment	SDC	Jovial, Assy	Successful Completion	Debug Hours
[Wolverton 74]	88	0.30	5.00		Military	TRW	Fortran, Assy	Object Code Words	Dollar Cost ($)
[Walston 77]	60	4.00	467.00	20.00	Scientific ; DP	IBM	Several	Y n Y Y Y Y	Y Y Y Y Y Y Y Mo
[Chrysler 78]	31				DP	SW House	Cobol	15 FP-like measures	n n Y Y n n n Hr
[Lawrence 81]	278	0.01	6.00		Data Proc	Australia	Cobol, +	Y n Y n n n	n n Y Y n n n Hr
[Boehm 81]	63	1.98	966.0		Various	TRW	Several	Y Y Y Y n Y	n Y Y Y n Y Y Mo
[Bailey 81]	18	2.10	100.8	28.80	Scientific	NASA	Fortran	Y Y Y Y n n	n Y Y Y n Y ? Hr
[Behrens 83]	25	22 FP	435 FP		Data Proc	Equitable Life	Cobol, 4GL	New FPs only	n n Y Y n ? ? Hr
[Thadani 84]	1	210.00	210.00		Data Proc	IBM	PL/I	Y n Y Y n n	n n Y n n n n Hr
[Jeffery 85]	103	0.02	9.80		Data Proc	Australia	Cobol, PL/I	Y n Y n n n	n n Y Y n n n Hr
[Card 87]	22	32.80	159.00	62.00	Scientific	NASA/ Goddard	Fortran	Y Y Y Y n n	n Y Y Y n Y ? Hr
[Gill 90]	65	0.22	128.00	15.60	Scientific	Aerospace	Fortran, Pascal	Y n Y n n n	Y Y Y Y Y n Y Hr
[Banker 91]	65	0.05 8 FP	31.00 616 FP	5.42 118 FP	Data Proc	Bank	Cobol	Y Y Y Y n n	Y Y Y Y Y Y Y Hr
[Kemayel 91]	200	n/a	n/a	n/a	Public Sector	Tunisia	Cobol	Y n Y Y n n	n n Y n n n n Mo

deemed a significant factor in the Federal Systems Division of IBM, which built large custom systems. The Software Engineering Laboratory of the University of Maryland has adopted the standard of a fixed 20% of reused code being counted as being "developed," and hence appearing in productivity measures (Bailey and Basili 1981) and (Card, McGarry, and Page 1987). Boehm (1981) offers a more complex, but relatively appealing approach in COCOMO. Given that the average project should spend 40% in design, 30% in coding, and 30% in testing, COCOMO calls for estimating the percentage of the original design, code, and testing effort that must be reproduced, and multiplying it by 0.4, 0.3, and 0.3, respectively, to get a weighted factor of the amount of work that must be reproduced. That factor is then multiplied with the old code size to get the "effective" number of lines developed by modifying the old code.

Note that one problem with using lines of code for the size estimate is that the final number of lines of code is not known at the start of the project, and so must be estimated. Albrecht and Gaffney (1983) proposed another size measure called *Function Points* (FPs), which counts the number of input files, output files, internal files, inquiries, and interface files. Function Points have been shown in some organizations to provide better predictors of effort than lines of code (Behrens 1983), and there is evidence that prediction equations based on FPs may be applicable between different organizations (Kemerer 1987). Some software productivity researchers have concluded that lines of code should be abandoned as a measure of software productivity output in favor of FPs (Jones 1991). Function Points are and will continue to be a fruitful area of software cost estimation model research.

Software managers generally divide software development projects into "phases" of requirements definition, functional design, detailed design, coding, unit test, and system integration/test. All of the software productivity studies surveyed measured the time that software engineers spend in the detailed design, code, and unit test phase. That still leaves several issues as to what other effort to measure:

1. How much of the "requirements definition" phase effort is measured, such as proposal preparation, user meetings, and background research?
2. How much of the "functional specification" phase is measured, e.g., the time spent by product planners or system engineers?
3. How much of "system testing" is measured? If the project is developed for an in-house customer, this effort is usually included. If the project is developed for an external customer, however, the customer may perform system testing that is not counted.
4. How much of the support staff effort is counted, including secretaries, building maintenance, personnel, and other general and administrative overhead?
5. How much of management effort is charged to the project, including reviews by senior managers, design-review time by non-project engineers, and similar items?
6. How much of the documentation effort is counted? In projects developed for the federal government, documentation is a significant portion of the effort.
7. How much of the engineers' "nonproductive" time is counted? Studies have shown that from 30% (Boehm 1981) to 50% (Brooks 1975) of software engineer time is not spent in activities that directly contribute to the final software product.

The "Productivity Input Measure" column of Table 11.1 indicates whether the study included the effort described in the preceding questions 1 through 6. As for measurements of nonproductive time, few studies indicate whether it was excluded. Table 11.1 indicates the granularity with which effort was measured. It is possible that studies that measured effort on the hour (Hr) level did not include non-productive hours. Studies that measured effort on the work-month level (Mo) are more likely to have included the nonproductive time.

Staffing Variables in Software Estimation Models

The distinguishing feature of the studies surveyed in this paper is that they included staffing variables as an explicit independent variable. In all cases, the models are attempting to account for differences in individual ability as a possible source of software productivity differences.

Individual differences: experimental results

In one of the earliest software productivity studies, Sackman, Erikson, and Grant (1968) measured the productivity of individual engineers solving the same problem with on-line terminals and off-line batch submissions. To their surprise, they noticed a 28:1 difference in the time between the fastest and slowest solution to the problem. This 28:1 ratio is widely and inaccurately cited as the range of individual programmer performance capability.

As clarified by Dickey (1981), the figure of 28:1 arose as follows: one subject programmed in assembly language in a batch mode and took 170 hours to solve an algebra problem. Another subject programmed in a high-level language on a time-shared system and took only six hours. The difference of 28:1 was caused partly by individual differences and partly by the deleterious effects of programming in a relatively time-inefficient language and in a time-inefficient manner. For individuals using the same language and the same system, the difference recorded by Sackman was only 5:1.

In response to Dickey, however, Curtis (1981) reaffirms individual differences by presenting another experiment in which three individuals took 39 minutes to find a planted bug and another individual took only three minutes, a 13:1 difference. (In fact, one individual never found the bug and quit after 69 minutes.) The Curtis study, however, is for a significantly smaller problem (minutes rather than hours).

The major unresolved issue for either of these studies is whether these ranges of performance in solving one small problem will be observed when the issue is scaled up to the full range of activities required of a practicing software engineer. Researchers cannot unequivocally generalize from the Sackman et al. data or the Curtis data that the long-term productivity of software engineers varies "on the order of a decimal order of magnitude." Furthermore, these small experiments do not strongly suggest that the same engineers will have similar productivity rates on different small problems.

In addition, it should be noted that controlled software experiments have also sometimes shown no significant correlation between performance and experience. Sheppard et al. (1979) found little correlation in small-scale experiments testing program comprehension and modification tasks. For debugging tasks, they found that programmers with less than 3 years experience showed correlation between debug-

ging time and 1, 2, or 3 years of experience, but they admit this may be a statistical artifact of the wider variation in times reported for this low-experience group. The correlation was not present for programmers with more than 3 years experience.

Of more interest would be studies of much larger, typical problems. Unfortunately, the cost of such an experiment is prohibitive, so published studies of individual productivity differences on large projects are confounded by differences in the tasks themselves. Thadhani reports one small study with six subjects working on a large (210 KLOC) program, each doing separate tasks. The range of productivity was 4:1, from 1200 to 300 LOC/MM. DeMarco and Lister (1985) report, on page 45, a "composite of the findings of three different sources" that show an individual performance difference of 2.5:1 from the best to the median performer, and 10:1 from best to worst. These range differences are supported by their "coding war games" exercise performed by over 600 software engineers.

Individual differences: field studies

Current software cost estimation models attempt to account for individual differences indirectly—by assuming that other measurable factors, such as experience, may substitute for personal productivity. Usually, the staffing variables included in a model are measures of the experience of an individual, as measured in years experience with various facets of the project. Some models also add measures of the capability of the individual. Banker, Datar, and Kemerer (1991), for instance, used the performance ranking of the individuals as a measure of capability. Other models used subjective management classifications of capability. Table 11.2 classifies the staffing variables used by published software cost estimation models into eight categories, and shows which categories are used by which model. The human factor categories are:

- Total experience—the total professional experience of the engineer or team
- Language experience—the experience of the engineer/team with the programming language for the project
- System experience—the experience with the operating system or development environment
- Application experience—the experience with the software application domain
- Hardware experience—the experience with the hardware platform
- Programmer capability—a measure of the capability of the individual or team
- Management capability—a measure of the capability of the management or systems analysts of the team
- Education—the level of formal education of the individual or an average for the team

A major problem of all of the studies with more than one developer is how to assign a measure for a multi-person team. Although simple averages of years of experience could be applied, it does not appear that any study actually did this. In most cases a subjective determination is made to classify the team into one of a limited

number of experience or ability categories. This is the approach, for instance, of both the COCOMO model (Boehm 1981) and the SPQR model of Jones (1986). Such arbitrary assignments, of course, diminish the repeatability of observed data and reduce the generalizability of the model.

Note that many popular cost estimation models, such as the SLIM model of Putnam (1978), do not require quantification of staffing variables per se. The SLIM model uses a "technology factor" constant to incorporate all factors other than project size and scheduled time. As such, it can be thought of as the correction factor "M." Putnam posits that this correction factor, or productivity index, stays within a narrow range for a given organization and changes relatively slowly over time. However, the SLIM database has observed productivity indexes for a single class of applications to vary by plus or minus 3 as a standard deviation. The difference between a productivity index of 12 and one of 6 is an effort estimation variation of as much as 4.23:1. Changing the index by even plus or minus one level changes the effort estimate by 60% (Mah and Putnam 1990). Therefore, the SLIM model allows for wide ranges in productivity.

Project team size

Programming-in-the-large is fundamentally a group activity. Weinberg (1971) was one of the first to recognize that improving psychological group dynamics would be likely to improve software productivity. He proposed "egoless programming" in order to improve group communication and effectiveness. Schneiderman (1980), in recognizing programming as a group process, advocates peer review and peer rating.

One of the easiest software metrics to measure is team size itself, and several researchers have investigated its impact on performance. The most famous anecdotal evidence is from Fred Brooks' experience on the OS/360 project documented in *The mythical man-month* (Brooks 1975). Brook's Law states that "Adding manpower to a late software project makes it later," due to increased communications and training efforts. Scott and Simmons (1975) documented the effect of increased communications due to different organizational styles, again showing how greater project communication requirements reduced productivity.

A thorough analysis of group size impact on productivity appears in the Conte, Dunsmore, and Shen (1986) textbook. They propose a Cooperative Programming Model (COPMO) developed by Thebaut and the authors at Purdue (1983). The generalized COPMO model is of the form

$$\text{Effort} = a + b \times \text{Size} + c \times (\text{Pbar})^d$$

where Pbar is the average team size, defined as E÷T, where E is effort in person-months and T is project duration in months. The Conte et al. textbook includes six databases of project size, effort, and time. They used these data to estimate a to be zero and d to be 1.5. They calculated a set of b and c parameters in b_i and c_i pairs for a set of "effort complexity classes." Using COCOMO's a priori modification factors, they defined four complexity classes and found the b_i and c_i for each class that minimize the Magnitude of Relative Error (MRE) measure for the COPMO model. With four such complexity class partitions, the COPMO model has an MRE of 25%, pro-

ducing an effort estimate within 25% of the actual effort for 75% of the COCOMO database projects. This level of accuracy is consistent with best performance for current software cost estimation models.

Staffing Variables Impact on Software Cost Model Estimates

Human factors have had mixed success in improving software model accuracy. Studies that have successfully shown that more-experienced teams were more productive include those by Chrysler (1978), Walston and Felix (1977), Thadhani (1984), and Card, McGarry, and Page (1987).

Chrysler tested five programmer-related variables, including age and various types of experience, and found higher levels to all be associated with a reduction in the number of hours required to produce COBOL programs. Of the five, the single most useful predictor variable was "experience at this facility."

Walston and Felix collected data on a set of 60 projects by IBM's Federal system division in the 1972–1976 time frame. First of all, they calculated the productivity, in lines of code per work-month (LOC/WM) for each project. They collected data by a questionnaire on some 68 variables concerning the development environment, nature of the problem, and staffing variables. Of the 68 variables, 29 showed significant spread of productivity between the upper third and lower third of the variable ranges. Five of the 29 variables concerned staffing variables, and the results are shown in Table 11.2.

Thadhani measured the productivity of six programmers working on a large PL/I program. Although his primary interest was the effect of response time on productivity, he noted in his sample that the two experienced programmers produced at a higher rate than the four less-experienced programmers. The two experienced programmers produced 1200 and 900 LOC/MM, while the less-experienced programmers produced 600, 600, 400, and 300 LOC/MM. Such a study was too small to have a high degree of statistical significance, and as mentioned earlier, may have been confounded by the differences in the tasks performed by the engineers.

Card et al. measured a set of eight "technology use" variables (e.g., structured programming) and 12 "non-technology" variables (e.g., programmer effectiveness) along with LOC/MM productivity for 22 projects at NASA's Goddard Space Flight Center. The "programmer effectiveness" measure was "a weighted measure of the development team's general and application-specific years of experience." They found that this programmer effectiveness variable had the highest positive correlation with productivity. A variable called "computer use," which was computer time per line of code, had a slightly higher negative correlation, probably because projects in trouble had to spend more computer time. Together, these two variables accounted for 54% of the variation in productivity.

An ANCOVA analysis of the remaining variables showed that no other variable accounted for more than 5% additional explanation of variance. They also found that the programmer effectiveness variable had the highest correlation to software reliability. The range of staff application experience (2.9–5.0 years) and overall experience (7.0–11.0 years) appears to be low, and it is not clear to what extent the transformation to a "programmer effectiveness" measure affects the significance of the results.

TABLE 11.2 Human factors in surveyed studies.

Study	Staffing Variables	Results
	T: Total Experience L: Language Experience S: System (OS) Exp. A: Application Exp. H: Hardware Exp. P: Programmer ability M: Manager Ability E: Education	
	T L S A H P M E	
[Sackman 68]	Y Y Y	High individual differences (28:1) in debug time, but confounded with different languages and on-line/off-line. Maximum 5:1 difference with same treatment.
[Wolverton 74]	Y	No correlation of programmer total experience with dollar cost of object code produced.
[Walston 77]	Y Y Y Y	3:1 variation in productivity between extremes of five human factor variables.
[Chrysler 78]	Y Y Y Y Y	Reductions in programming time correlated with 5 different experience related variables. "Experience at this facility" most useful single predictor.
[Lawrence 81]	Y	No increase in productivity after the first year of experience.
[Boehm 81]	Y Y Y Y Y	Human factors are 5 of 15 factors which together reduce prediction mean relative error from 60% to 19%.
[Bailey 81]	Y Y Y Y Y	Human factors are 5 of 21 factors tested. Minimum model prediction error obtained without using any of them.
[Behrens 83]	Y	Experience factor did not improve model prediction error.
[Thadani 84]	Y Y	Sample of 6 programmers on real-world application which showed 3:1 productivity differences between experienced an inexperienced engineers.
[Jeffery 85]	Y Y	Follow-up to [Lawrence 81] with same results: no productivity increase after first year of experience.
[Card 87]	Y Y	Experience had highest correlation to productivity (.53) of 12 non-technology and 8 technology factors.
[Gill 90]	Y Y	Slight decrease of productivity with increased fraction of project time by highly experienced personnel. (Task complexity not controlled for.)
[Banker 91]	Y Y	Ability and application experience not significant at usual levels after controlling for other factors such as project size.
[Kemayel 91]	Y Y Y Y Y	Two of five experience variables significantly correlated with productivity, but together explained only 7.5% of variance.

Other studies have failed to find significant effects of differences in experience level. An early study by Wolverton (1974) failed to find any correlation between experience and programming cost per object instruction. Lawrence (1981), and Jeffrey and Lawrence (1985) conducted two studies of over 350 programs prepared by 17

Australian organizations in 1976 and 1980. Their data showed very slight increases in productivity after the first year of COBOL programmer's experience, but no improvement in subsequent years. In general, they conclude (on page 55) by noting "a confusing relationship between education, experience, and productivity suggesting no clear trend of increasing productivity with education and experience." This confirms the experimental work by Sheppard et al. (1979), also showing early improvement with experience but no significant differences after the first few years.

Kemayel, Mili, and Ouederni (1991) studied the productivity of 200 individual Tunisian programmers, collecting data on 33 variables categorized as "personnel," "process," and "user-related" factors. Of the 33 variables, five were staffing factors—education, applications domain experience, programming language experience, virtual machine experience, and experience with the user community. Of these, only the last two were statistically significant, and these variables combined explained only 7.5% of the variation in productivity. They found very wide variation (15:1) in KDSI/MM productivity ratings, which they attribute to Tunisia's practice of tenured employment of even low-skill programmers.

One limitation of most of the studies cited above is that the researchers have done simple correlations of (typically) one experience variable and, for example, productivity. Such correlations might mask the effect of other variables that might be correlated with experience but are not controlled for in the model. Of the research surveyed here, Banker, Datar, and Kemerer (1991) reported the effect on the standard error when the human factor variable was removed.

They considered two variables: "No Experienced" (a dummy variable equal to 1 when no team member exceeded 2 years experience) and "Top Staff" (the percentage of project hours charged by individuals rated above average in the organization's performance review system). The "No Experienced" variable, and the "Top Staff" variable were not significant at usual levels ($p = 0.41$ and $p = 0.11$).

Another way to investigate the value added by the staffing variables in current cost estimation models is to examine the relative performance of these models and to determine factors common to the better-performing models. Interestingly, such an analysis suggests that the best software effort prediction models do not require the use of staffing variables as an independent variable. Of the models surveyed, the best results in terms of predicting the actual project values were obtained by Bailey and Basili (1981), using a baseline model and correction factors for the NASA Goddard data set.

Although they collected staffing variables data, they found that the correction factors for project complexity and development methodology alone were sufficient to build a model with a standard multiplicative error of only 16%. Conte, Dunsmore, & Shen (1986) also suggests a model (COPMO) that uses both estimated project size and team size, with modification factors only for project complexity, that achieves a mean absolute relative error of 21%.

From the above review it can be concluded that staffing variables, as currently operationalized, have not been shown to be a particularly useful addition in software cost estimation models. After accounting for measures of the project size, and modifying this estimate for project complexity differences and tool/methodology differences, additional correction for human performance differences appear to not significantly improve model accuracy. *Given that this result is at odds with what*

practicing software managers believe about factors affecting project performance, this suggests that more work is necessary to improve the modeling of staffing factors.

Future Research and Recommendations

The main difficulty with incorporating staffing variables in software cost models is that it is not well understood why individual and team programming performance varies. "We do not know if these differences are attributable to differences among individuals' speed of information processing while programming, the nature of the information processing themselves, or some combination of the two." (Laughery and Laughery 1985). Weinberg (1971) suggests that differences in performance, especially debugging speed, are probably due to the exact structure of the problem itself, and whether a similar exact problem has been seen before. If debugging time is related to specific pattern matching, then studies of ranges of different times to debug a small problem are not useful: the variation is only due to the variation of particular previously-solved problems. Supporting this difference is the failure of almost all studies to show a correlation between tests of programming skill and actual programming performance.

The most commonly accepted theories for accounting for individual differences are the cognitive psychology theories of how programming knowledge is represented. These theories are surveyed in Laughery and Laughery (1985), and Curtis (1990), and are only briefly described here.

Newell and Simon introduced a major theory of human problem solving via goal-seeking production rules, today the basis of knowledge-based artificial intelligence systems. Brooks (1977) applied the Newell and Simon theory to the programming task by studying in detail a single expert programmer. He identified 23 protocols used to produce code, themselves producing the 73 actions that generated the simple program under development.

He estimated that the total number of rules used by an expert programmer was in the tens of thousands, consistent with the 31,000 rules estimated used by a chess expert. He argues that domain knowledge (the set of rules mastered), determines differences in programming skill. Schneiderman and Mayer (1979) propose that syntactic and semantic knowledge is stored separately, so that, for instance, the algorithm for finding the largest integer in an array is stored independent of the form of the FORTRAN program to implement it.

Program comprehension is a central skill for software maintenance and an important component of original software design. Robson et al. (1991) recently surveyed the various psychological research on how programs are comprehended. They cite 48 papers studying the effectiveness of code artifacts on comprehension and various other theories of program comprehension. As an example, Letovsky (1987) analyzed the protocols employed by maintenance programmers in reading and understanding programs, characterizing the process as an iteration of questions and conjectures. He classifies these processes into a taxonomy of psychological constructs including: slot filling, abductive reasoning, symbolic evaluation, discourse rules, generic plans, and endorsement rules.

Brooks (1983) attributes individual differences in program comprehension performance to differences in programming knowledge, problem-domain knowledge, and hypothesis-confirmation search strategies. Soloway and Ehrlich (1984) identify two distinguishing components of expert programming skill to be programming "plans" or knowledge schemas and "rules of programming discourse," or useful coding conventions.

It seems apparent from the above that cognitive psychology offers a rich, complex, and growing set of theories to account for the observed differences in individual performance. Expert programmers may be those who have learned rules appropriate to the problem under examination. This includes semantic and syntactic knowledge of the computer language and computing environment, and often domain-specific knowledge as well. Vast individual differences in individuals may swamp any observed variation due to different treatments in programming experiments and software productivity studies. In an excellent critical review of programming psychology experiments, Sheil (1981) expresses the difficulty of measuring productivity differences as follows:

> However programmers' knowledge bases are actually organized, their existence and size seems clear. Hypotheses which posit differences in either individual aptitude or task difficulty are therefore, at best, extremely difficult to investigate, as the enormous size of the knowledge bases being drawn on imply that different individuals approach the same task with vastly different resources. Comparing their behavior is like contrasting the work of two tile layers, each of whom has covered an equivalent wall, working with radically difference sizes and colors of tiles. Similarity of pattern is unlikely.

Significant difficulties remain in measuring the richness of an individual's knowledge base and its conceptual fit to a problem's domain knowledge. Gibson and Senn (1989) for instance, found that programmers performing software maintenance tasks cannot reliably separate the syntactical complexity of a program's text from the semantic complexity of the modification task itself. Psychological constructs that cannot be distinguished cannot be measured.

Cognitive psychology offers a path for understanding why increases in, say, program size increase programming effort. It offers a diverse set of theories from which hypotheses can be drawn, measures defined, and experiments conducted. Vessey and Weber (1984) chide the computer science community for concentrating research on predicting dependent variable effects of structured programming, rather than understanding those effects. Their criticism applies to the entirety of software cost estimation model research.

Given the richness and complexity of the still-early understanding of software cognitive psychology, staffing variable measures like "years of application experience" are at best crude approximations of the differences among individuals. In addition, they do not at all address the issue of coordinating the mix of experiences and aptitudes of multiple agents in software development teams. It is, therefore, not surprising that such measures often fail to significantly improve the accuracy of software cost estimation models.

What are needed are more well-grounded variables that reflect these more complex notions of individual differences, most especially the staff/problem fit issue.

Author Biography

Chris F. Kemerer is the Douglas Drane Career Development Associate Professor of Information Technology and Management at the MIT Sloan School of Management. He received the B.S. degree, magna cum laude, in economics and decision sciences from the Wharton School of the University of Pennsylvania and the Ph.D. degree from the Graduate School of Industrial Administration at Carnegie Mellon University.

Prior to his graduate studies he was a Principal at American Management Systems, Inc., the software services firm. His research interests are in the measurement and modeling of software development for improved performance, and he has previously published articles on these topics in leading academic journals, including *Communications of the ACM*, *IEEE Transactions on Software Engineering*, *Information and Software Technology*, and *Management Science*. Dr. Kemerer serves on the editorial boards of the *Communications of the ACM*, *Information Systems Research*, *The Journal of Organizational Computing*, *The Journal of Software Quality*, and *MIS Quarterly*, and is a member of the IEEE Computer Society, ACM and The Institute for Management Sciences.

Michael W. Patrick graduated from MIT with a BS and MS in Computer Science in 1980. He was the chief architect at Texas Instruments for the IBM Token Ring Chip Set, and has served on the IEEE 802.5 committee for local area networks. He has served as Research Associate at Ztel, Engineering Manager at General Computer Corporation, and Vice President of Engineering at Cryptall Communications Corp. He is currently a graduate student in the Management of Technology program in the Sloan School of Management at MIT.

Works Cited

Albrecht, A.J. and J. Gaffney. 1983. Software function, source lines of code, and development effort prediction: A software science validation. *IEEE transactions on software engineering*. SE-9 (6): 639–648.

Bailey, J.W. and V.R. Basili. 1981. A meta-model for software development resource expenditures. *Proceedings of the 5th international conference on software engineering*. 107–116.

Banker, R.D., S.M. Datar, and C.F. Kemerer. 1991. A model to evaluate variables impacting productivity on software maintenance projects. *Management science*. 37 (1): 1–18.

Behrens, C.A. 1983. Measuring the productivity of computer systems development activities with Function Points. *IEEE transactions on software engineering*. SE-9 (6): 648–652.

Boehm, B. 1981. *Software engineering economics*. Englewood Cliffs, N.J.: Prentice-Hall.

Brooks, F. 1975. *The mythical man-month*. Addison-Wesley.

Brooks, R. 1977. Towards a theory of cognitive processes in computer programming. *International journal of man-machine studies*. 9: 737–751.

Brooks, R. 1983. Towards a theory of the comprehension of computer programs. *International journal of man-machine studies*. 18: 543–554.

Card, D.N., F.E. McGarry, and G.T. Page. 1987. Evaluating software engineering technologies. *IEEE transactions on software engineering*. SE-13 (7): 845–851.

Chrysler, E. 1978. Some basic determinants of computer programming productivity. *Communications of the ACM*. 21 (6): 472–483.

Conte, S.D., H.E. Dunsmore, and V.Y. Shen. 1986. *Software engineering metrics and models*. Reading, Ma.: Benjamin-Cummings.

Curtis, B. 1981. Substantiating programmer variability. *Proceedings of the IEEE*. 69 (7): 846.

Curtis, B. 1990. Fifteen years of psychology in software engineering: Individual differences and cognitive science. In *Software state of the art: Selected papers*. ed. DeMarco T. and T. Lister. New York, N.Y.: Dorset House.

DeMarco, T. and T. Lister. 1987. *Peopleware*. New York, N.Y.: Dorset House.

Dickey, T.E. 1981. Programmer variability. *Proceedings of the IEEE*. 69 (7): 844–845.

Gibson, V.R. and J.A. Senn. 1989. System structure and software maintenance performance. *Communications of the ACM*. 32 (3): 347–358.

Heemstra, F.J. and R. Kuster. 1991. *Controlling software development costs: A field study*. University of Technology.

Jeffery, D.R. and M.J. Lawrence. 1985. Managing programming productivity. *Journal of systems and software*. 5: 49–58.

Jenkins, A.M., J.D. Naumann, and J.C. Wetherbe. 1984. Empirical investigation of systems development practices and results. *Information management*. 7: 73–82.

Jones, C. 1986. *Programming productivity*. New York, N.Y.: McGraw-Hill.

Jones, C. 1991. *Applied software measurement*. New York, N.Y.: McGraw-Hill.

Kemayel, L., A. Mili, and I. Ouederni. 1991. Controllable factors for programmer productivity: A statistical study. *Journal of systems and software*. 16: 151–163.

Kemerer, C.F. 1987. An empirical validation of software cost estimation models. *Communications of the ACM*. 30 (5): 416–429.

Kemerer, C.F. 1991. Software cost estimation models. In *Software engineers reference book*. ed. McDermid, J. Oxford, England, U.K.: Butterworth-Heinemann Ltd.

Laughery, K. and K. Laughery. 1985. Human factors in software engineering: A review of the literature. *Journal of systems and software*. 5: 3–14.

Lawrence, M.J. 1981. Programming methodology, organizational environment, and programming productivity. *Journal of systems and software*. 2: 257–269.

Letovsky, S. 1987. Cognitive processes in program comprehension. *Journal of systems and software*. 7 (4): 325–339.

Mah, M.C. and L.H. Putnam. 1990. Is there a real measure for software productivity? *Programmer's update*. 26–36.

Phan, D., D. Vogel, and J. Nunamaker. 1988. The search for perfect project manage ment. *Computerworld*. 95–100.

Putnam, L.H. 1978. A general empirical solution to the macro software sizing and estimating problem. *IEEE transactions on software engineering*. 345–361.

Robson, D.J., K.H. Bennett, B.J. Cornelius, and M. Munro. Approaches to program comprehension. *Journal of systems and software*. 14: 79–84.

Sackman, H., W.J. Erikson, and E.E. Grant. 1968. Exploratory experimental studies comparing online and offline programming performance. *Communications of the ACM*. 11 (1): 3–11.

Schneiderman, B. 1980. *Software psychology: Human factors in computer and information systems*. Cambridge, Ma.: Winthrop Press.

Schneiderman, B. and R.E. Mayer. 1979. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International journal of computer and information sciences*. 8: 219–238.

Scott, R.F. and D.B. Simmons. 1975. Predicting programming group productivity—a communications model. *IEEE transactions of software engineering*. SE-1 (4): 411–414.

Sheil, B.A. 1981. The psychological study of programming. *Computing surveys*. 13 (1).

Sheppard, S.B., B. Curtis, P. Milliman, M.A. Borst, and T. Love. 1979. First year results from a research program on human factors in software engineering. pp. 1021–1027. National Computer Conference. New York, N.Y.

Soloway, E. and K. Ehrlich. 1984. Empirical studies of programming knowledge. *IEEE transactions on software engineering*. SE-10 (5): 595–609.

Thebaut, S.M. 1983. *The saturation effect in large-scale software development: Its impact and control*. Ph.D. thesis. Purdue University Dept. of Computer Science.

van Genuchten, M. 1991. Why is software late? An empirical study of the reasons for delay in software development. *IEEE transactions on software engineering*. 17 (6): 582–590.

Vessey, I. and R. Weber. 1984. Research on structured programming: An empiricist's evaluation. *IEEE transactions on software engineering*. SE-10 (4): 394–407.

Walston, C.E. and C.P. Felix. 1977. A method of programming measurement and estimation. *IBM systems journal*. 16 (1): 54–73.

Weinberg, G.M. 1971. *The psychology of computer programming*. New York, N.Y.: Van Nostrand Reinhold.

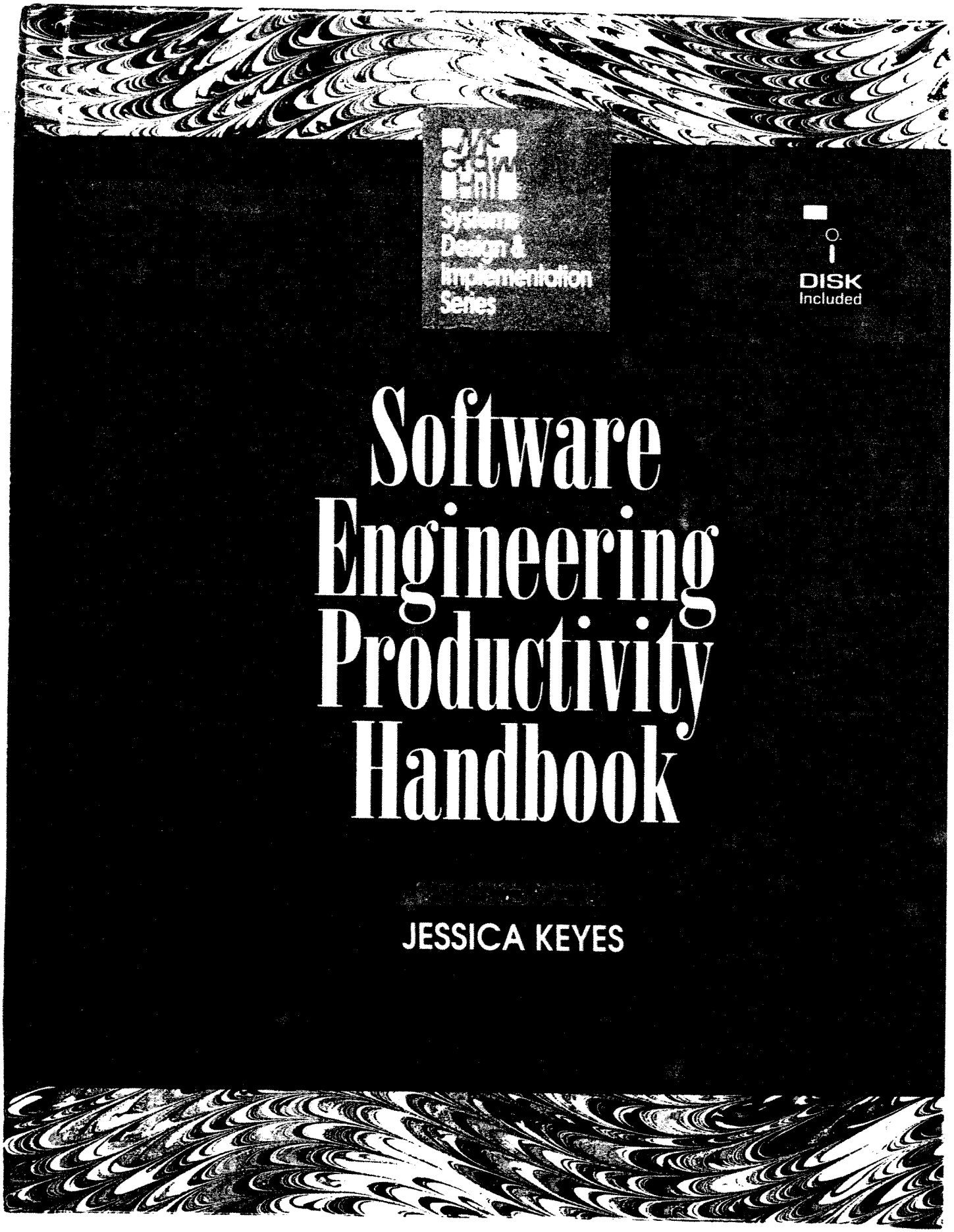
Wolverton, R.W. 1974. The cost of developing large-scale software. *IEEE transactions on computers*. C-23 (6): 615–636.

Software Engineering Productivity Handbook

Jessica Keyes, Editor

Windcrest®/McGraw-Hill

New York San Francisco Washington, D.C. Auckland Bogotá
Caracas Lisbon London Madrid Mexico City Milan
Montreal New Delhi San Juan Singapore
Sydney Tokyo Toronto


McGraw-Hill
Systems Design & Implementation Series
DISK
Included
Software Engineering Productivity Handbook
JESSICA KEYES